Responsible Disclosure is a Two-Way Street: Empirically Measuring the Responsible Disclosure Contract in the Firmware Ecosystem

Hui Jun Tay[1], Souradip Nath[1], Arvind S Raj[1], Abhay Bhat[1], Ishan Bansal[1], Audrey Dutcher[1], Moritz Schloegel[2], Adam Doupé[1], Tiffany Bao[1], Yan Shoshitaishvili[1], Ruoyu Wang[1]

[1]Arizona State University, [2]CISPA Helmholtz Center for Information Security
{*htay2, snath8, arvindsraj, abhat36, ibansal5, dutcher*}*@asu.edu*
{*moritz.schloegel*}*@cispa.de,* {*doupe, tbao, yans, fishw*}*@asu.edu*

***Abstract*—Responsible disclosure is the process by which researchers and vendors cooperate to release information on newly discovered vulnerabilities to the public in an ethically responsible manner. Proper vulnerability disclosure is especially important for the security of embedded firmware in Internet-of-Things, where a single exploit often impacts thousands of consumer devices.**

The current prevalent belief is that disclosing vulnerabilities on *some* devices is better than not bringing up observed vulnerabilities at all, leaving us with an unknown set of potentially affected devices. Implicitly, this assumes that the potential vulnerability impact of these "invisible" devices is minimal relative to the rest of the publicized set. Should this assumption prove false, a partial reporting of vulnerable devices would conversely pose a *greater security risk*, as malicious actors can trivially use released exploits to target the invisible devices.

In this paper, we seek to quantify the degree to which such vulnerable devices are overlooked during responsible disclosure. We provide a lower-bound estimate of the security impact these "invisible" yet vulnerable devices have for end-users. To this end, we model the disclosure process and develop an automated pipeline, *BucketLeak*, to run a collection of 54 vulnerability exploitation scripts from the years 2010–2025 against a large-scale dataset of 3,569 firmware images belonging to 566 router and camera devices. Our pipeline uncovers 467 unique device-exploit pairs (DevExPairs), of which 422 are undisclosed potential N-days that correspond to 290 device models still in circulation. Furthermore, 181 of the models with undisclosed vulnerabilities are still vulnerable even with the latest versions of their firmware installed. By scanning the Internet-of-Things with ZoomEye, we find that these 181 vulnerable yet undisclosed devices have more than 1.04 million real-world device counterparts still active and discoverable over the public Internet.

1. Introduction

The debate over proper handling of vulnerability knowledge has raged from as early as the 19th century in the discussion of locks and lockpicking [30] to modern day cybersecurity systems. Today, the consensus is that disclosing found vulnerabilities is an ethical imperative[1] and even required by academic conferences [1], [32], [40], [58]. Generally, researchers are expected to engage in a process called *responsible disclosure*[2].

> *"Ne craindra-t-on pas que nous ne donnions en même temps des leçons aux voleurs."*
>
> "But is there not this danger, that at the same time we shall be giving lessons to the thieves?"
>
> —M. de Réaumur (French original), translated by A. C. Hobbs [30]

Based on a guide from the US Computer Emergency Response Team Coordination Center (CERT/CC) [31], this process can be modeled as follows. *Researchers* discover a vulnerability and disclose it to the respective *developer(s)*, keeping the details private for a certain amount of time, such as 45 days (with 90 days being a common industry practice). After this time, they should notify the public, i.e., the *end-users* of the vulnerable software or device, of the vulnerability regardless of whether a patch was released. Ideally, however, the developer(s) create a patch (often with the help of the *researcher* who may verify that the vulnerability is mitigated) as soon as possible and then publicly release it alongside information on the vulnerability. End-users should monitor these announcements and, based on information about the vulnerability, organizational risk, availability, or testing of the patch, apply the patch or employ other mitigations. Note that all entities in this model can be companies.

While the responsible disclosure process can apply to any computer security field, it is particularly pronounced when discussing embedded security for IoT devices. As embedded devices are designed to be sold in large units at low cost, security is rarely a priority for their vendors [23], [47], [59]. Unsurprisingly, this has led to IoT researchers

1. Edge cases exist and require careful consideration before action [34].
2. Also called *coordinated disclosure*, although the "charged" term *responsible* has seemingly won out, at least in academic circles.

performing fuzzing [19], [49], [51], [60], [61] or program analysis [27], [33], [48] reporting dozens or even hundreds of vulnerabilities per paper. In contrast, IoT vendors have been comparatively slow in adopting the responsible disclosure processes [23], [59]. Given the highly heterogeneous nature of embedded system firmware and sheer number of devices they operate on [11], [47], having a well-developed responsible disclosure process becomes increasingly relevant as researchers in the field are discovering more IoT vulnerabilities with each passing day.

Meanwhile, concerns about the responsible disclosure process have led to research assessing its effectiveness [7], [8], [17], ethics [23], [34], [38], and impact [2], [37]. As commonly experienced in practice and documented by these works, the disclosure process in the real world often suffers from many issues ranging from the timeliness of patches to developer unresponsiveness to challenges in communicating the severity of a vulnerability and others. However, all our research and expectations of research are grounded in the belief that even this flawed vulnerability disclosure process is better than none, as it at least provides some mitigation to end-users.

For the responsible disclosure process to work, we find that it has the following implicit contract (which we make explicit in Section 3):

- All vulnerable software versions are acknowledged by the vendors (so that end-users can either mitigate or apply patches).
- All vulnerable software versions are patched by the vendors.
- All vulnerable software versions are either (1) patched correctly or (2) acknowledged by the vendors if they are not going to patch.

In this paper, we re-examine the validity of this contract *in practice* by conducting the first context-refined large-scale study of vulnerabilities in the embedded firmware of real-world routers and camera devices. We specifically select embedded devices where a broad diversity of devices, little oversight, fast turnaround times, and a highly exposed nature—routers are our gateway to the Internet and cameras have significant privacy concerns attached—create a relevant ecosystem in which any violations of this contract could have significant consequences for end-users.

To conduct our study, we sample 54 exploits and run them against 3,569 rehosted firmware images corresponding to 566 devices from eight vendors. We discover 467 device-exploit pairs, each a potential N-day for that device. Then, we compare this empirically verified set of exploitable devices against those reported in the corresponding responsible disclosure processes. Based on these results, we show that this contract is systematically violated in practice.

While the situation is quite nuanced, our most impactful finding is that *many* publicly reported vulnerabilities actually exist in *more* devices and firmware images than reported by vendors. Further compounding the issue is that we identify these with publicly available exploits.

Therefore, we identify a *hidden risk* of the responsible disclosure process: As the responsible disclosure process often leads to publication of proof-of-concept exploits or details on the vulnerabilities, malicious actors can easily weaponize this public knowledge to attack *all* devices affected by it, even those not acknowledged via the responsible disclosure process. End-users, however, retain only a limited view of the devices affected by this new vulnerability, lulling them into a false sense of security. Essentially, this (unintended) failure to disclose *all* vulnerable devices creates an *invisible gap* between the actual and perceived impact of a disclosed vulnerability.

Our work is the first to study this gap and quantify the impact thereof. Our data provides evidence that up to *twice as many* devices have vulnerabilities that fall into this gap compared to devices that are disclosed. We find that this gap comprises over 1.71 million active real-world devices, of which 1.04 million map to devices we verified are still vulnerable in their latest firmware versions, i.e., no patch is available. This gap is the result of 422 undisclosed N-days, of which we verified 260 can still exploit the latest device versions.

In providing these statistics and examining their sources, we hope to highlight this hidden risk of the responsible disclosure process so that the community can continue the disclosure discussion and chart a path forward that balances all stakeholder goals in light of these findings. Note that this work does not advocate for either non-disclosure of vulnerabilities or full-disclosure of vulnerabilities. Rather, it cautions that responsible disclosure is neither a process that can be carried out on autopilot nor a "fire and forget" task: It requires careful consideration on both sides of the street.

Contributions. In summary, our contributions are:

- A methodology for discovering the ground truth of vulnerable firmware images, patched firmware images, and end-of-life devices from a set of devices/firmware and vulnerabilities.
- An application of this methodology to 3,569 firmware images across 566 devices and 54 exploits.
- Identification of 422 undisclosed N-days, of which 260 were empirically confirmed to affect the latest device version.

We will release all research artifacts and data from our paper upon publication.

2. Background and Motivation

The following sections provide definitions and background knowledge for the responsible disclosure process as applied to vulnerabilities discovered in embedded devices.

2.1. Responsible Disclosure Process

Ethical handling of discovered vulnerabilities requires the disclosure of their information to the respective developers. CERT/CC standards [31] recommend that security researchers should only publicly release details of the vulnerability if an appropriate patch and/or official announcement

is not made by a given deadline. Many developers, including most major vendors of routers and other embedded devices, have procedures in place for vulnerability researchers to report any discovered vulnerabilities in a coordinated way [6], [9], [22], [36], [42], [57], [63]. Some administer their own vulnerability disclosure programs and/or work with third-party platforms, such as Bugcrowd [14]. While instructions for reporting bugs to these programs are usually well documented on their official sites, little information on the next steps (such as verification and/or remediation of a vulnerability) is publicly available. Furthermore, some vendors only issue advisories for specific categories of security vulnerabilities, not all that are reported. TP-Link, for example, only announces vulnerabilities that are rated CRITICAL by its internal team [57]. Anything lower will not have a security advisory issued. As such, we are only able to observe the inputs and outputs of the vendor side of the process for the purposes of our analysis.

We thus model the responsible disclosure process of a newly discovered vulnerability as a series of steps based on CERT/CC guidelines and common developer vulnerability disclosure procedures [6], [12], [13], [42], [57].

1) A researcher *discovers* a new vulnerability for one or more devices and device firmware versions.
2) The researcher *reports* the vulnerability and a (potentially incomplete) list of affected devices to the appropriate vendor(s).
3) The vendor analyzes all codebases to enumerate the vulnerability over potentially affected device models, *verifies* the reported vulnerability, and creates *patches* to mitigate it on affected devices.
4) The vendor coordinates with the researcher to publicly *disclose* the vulnerability and release patches.
5) End-users *patch*, *apply mitigations*, or *retire* affected devices accordingly.

However, in practice, not all vulnerable devices affected by a new exploit are identified, disclosed, or patched. In Steps 1 and 2, the researcher is unlikely to be able to discover or report all affected devices in their initial investigation. Steps 3 and 4 depend on the vendor's willingness to address these vulnerability reports, which prior works have noted to be notoriously unreliable [11], [47]. By the time the list of affected devices reaches Step 5, many devices are likely to have been *omitted* from the disclosed list. End-users, however, have no knowledge of the processes from Steps 1 to 4. Without clarification, they cannot know that the released list is not complete, thus diminishing their security. We model this study of a non-ideal disclosure process as a *Bifurcation Pipeline*, showing how knowledge of a vulnerability and its affected devices is diluted while it passes through the disclosure process.

2.2. Vulnerability Discovery with Rehosting

Recently, researchers performing vulnerability discovery on embedded devices have made increasing use of firmware rehosting techniques to scale-up their efforts [18], [33], [50], [52], [60]. These approaches find and report up to *hundreds* of exploits as part of their research, many of which are shared with vendors via the responsible disclosure process.

Firmware rehosting is a technique that enables dynamic analysis of firmware images without the corresponding device hardware. This is often achieved by "running" firmware images in an emulator that enables specific instrumentation of its programs, a key requirement for the use of automated vulnerability discovery techniques. This scales the analysis of firmware images on commodity hardware.

There are two commonly-known approaches to firmware rehosting, *full-system rehosting* and *selective rehosting* [25], [52]. Full-system rehosting aims to achieve a fully accurate emulation of the entire firmware image similar to running the image on the corresponding device hardware. While this high fidelity emulation enables accurate and thorough analysis of the firmware image, it is often difficult to achieve due to challenges of emulating the underlying device behavior and dealing with proprietary hardware accurately. However, selective rehosting emulates only the executables and filesystem of the firmware related to the targeted service (e.g., the web interface or a network service). This limits high fidelity emulation to selected firmware services while ignoring the rest, but is easier to achieve [52], [60].

With a rehosted target, researchers can perform dynamic analysis through exploit replay using platforms such as Metasploit [54] and routersploit [55]. Often, vendors request a proof-of-concept as part of their vulnerability reporting procedure [6], [15], such as a script or payload that can be used to demonstrate and verify the vulnerability. In this work, we examine exploit scripts from Metasploit [54], routersploit [55], and other third-party websites such as exploit-db [46] and GitHub [28].

2.3. Terminology

We use the following terminology throughout this paper when referring to embedded devices and their firmware.

Device refers to a physical router or camera, which may support multiple *firmware versions* over its lifetime. Each version identifies the type of firmware code running in the actual device or emulator, known as the *firmware image*.

Device lifecycle refers to the time period where a device is in use and supported by their vendor, between when the device is first released, to when it reaches obsolescence or End-of-Life (EOL) [11]. During the device lifecycle, the device should continue to receive patches from its vendor.

Firmware version refers to the identifier (e.g., version number or build code) assigned to an iteration of a firmware image by its developer. As our dataset uses vendor firmware for devices, we consider the developer to be the vendor for our analysis. Later versions represent newer releases that incorporate vendor updates, fixes, or feature enhancements. In the rest of this paper, a firmware version is one-to-one mapped to a firmware image.

An *exploit* is a proof-of-concept script that takes advantage of an existing vulnerability to trigger a security issue.

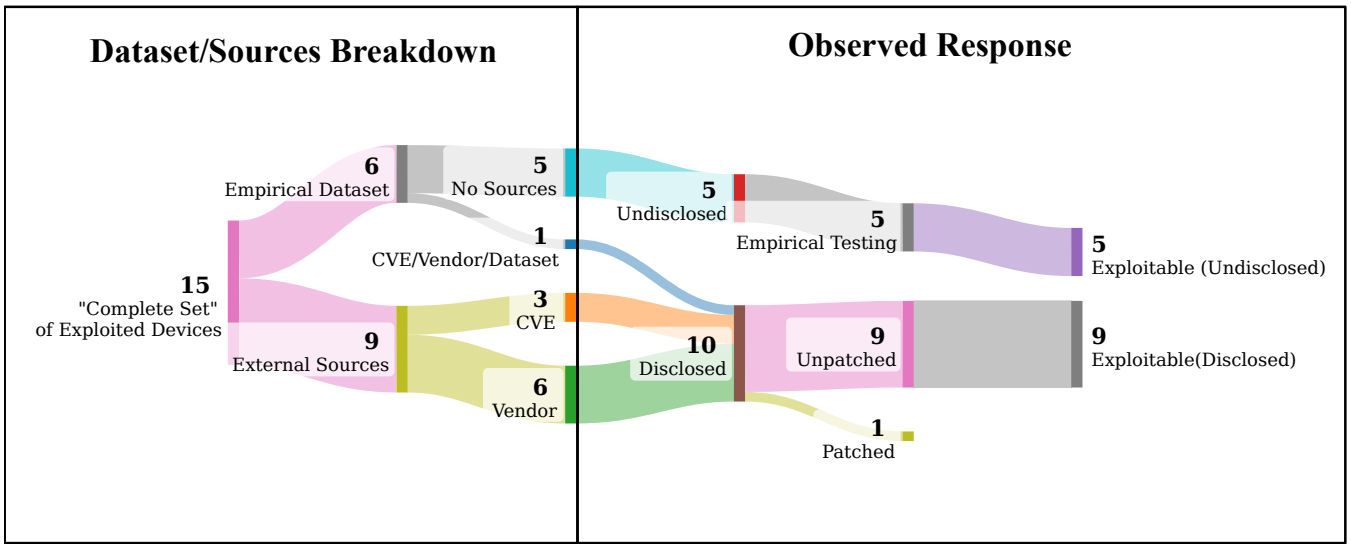

Figure 1: Pipeline of the responsible disclosure process for devices affected by CVE-2017-6334.

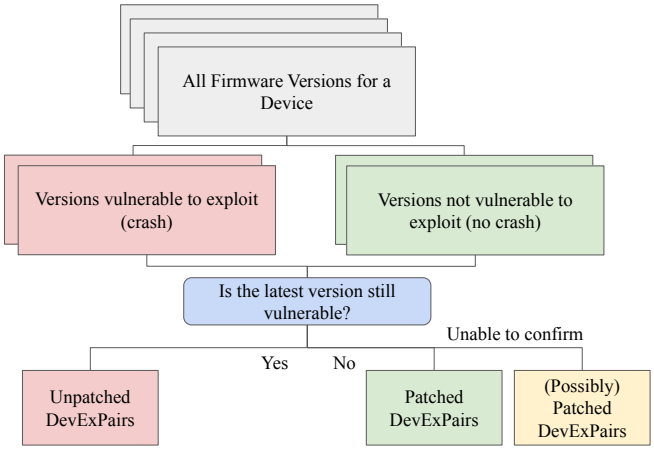

Figure 2: Process to categorize firmware versions, devices, and exploits into patched and unpatched groups.

An exploit affects *one or more* firmware versions for a single device. It may also work on multiple devices, even when these devices belong to different vendors.

A *Device-Exploit Pair (DevExPair)* refers to the unique pair of a vulnerable device and an exploit that triggers one vulnerability. Using DevExPairs allows us to condense crashes that are duplicates of the same vulnerability across multiple (older) firmware versions into one entry. We further categorize the device as *patched* or *unpatched* depending on if the latest firmware version is vulnerable or not. The exact decision tree is outlined in Figure 2.

An example. We rehost a *firmware image* for the router *device* DCS-930L. The *firmware version* of this image is *DCS_930L_REVB_FIRMWARE_2.00B6*. CVE-2019-10999 describes a vulnerability in DCS-930 versions below 2.16.01 with a link to a GitHub repository detailing a proof-of-concept script. We use this information to create an *exploit* that we run against the emulated firmware image. This exploit successfully triggers the vulnerability in our emulation, and it may potentially affect other versions as listed in the CVE. We group all exploited firmware versions under a single *DevExPair*, CVE-2019-10999//DCS-930L, which is considered a *patched* DevExPair, as it does not affect the latest version of firmware (cf. Figure 2).

2.4. Motivating Example: Netgear CVE-2017-6334

Realistically, not all vulnerabilities reported will receive patches or be known to the public. Researchers are limited by their lack of time, resources, and access to proprietary devices. Vendors must choose how to prioritize their support budget as more devices are released over time. As a result of these limitations, DevExPairs are overlooked or neglected throughout the disclosure process, resulting in a smaller pool of vulnerable devices being disclosed to the end-user. We model this non-ideal disclosure process as a **Bifurcation Pipeline**, showing how knowledge of a vulnerability and its affected devices is diluted while it passes through the disclosure process.

Figure 1 shows the number of unique Netgear devices passing through the Bifurcation Pipeline for CVE-2017-6334. Our testing finds six Netgear devices in our dataset that are vulnerable to the exploit, only one of which matches a device reported in the CVE description (DGN2200v1). We combine this set of devices with the three remaining DGN2200 devices mentioned in the original vulnerability report as well as another six devices listed on the security advisory announced by Netgear. The resulting set of 15 devices represents the complete set for the purposes of this example. This set consists of a mix of devices that we could empirically test and exploit, devices that are reported to be vulnerable but that we could not exploit, and devices reported to be vulnerable that did not have a publicly

available firmware image for us to test. This is the best approximation of the complete set we can hope to achieve. We now consider the disclosure pipeline.

In Step 1, a researcher discovers the vulnerability for one device they have access to. Researchers often have limited access to devices and their corresponding firmware for testing. Here, the researcher had access to at least four devices, which they considered vulnerable and reported. As the vendor, Netgear may acknowledge or dismiss any of these devices for various reasons. For CVE-2017-6334, only one of the four devices originally reported was acknowledged by Netgear [20], while the other three were announced as not vulnerable. This is reflected in Figure 1 under Dataset/Sources Breakdown, showing how three devices were dismissed. The disclosure suggests that Netgear's internal testing confirmed that the vulnerability exists on six other devices. At this point, seven devices are confirmed vulnerable. These seven vulnerable devices were acknowledged in the vendor disclosure as part of the support announcement, yet only one device (DGN2200v1) received a patch. The remaining six were not patched because Netgear considered them End-of-Life (EOL). From this case study, we see that out of the original set of 15 affected devices, only ten were mentioned in the final disclosure, and only one out of 15 devices was patched.

Analyzing the remaining 14 unpatched devices, we see that five were omitted or overlooked from the entire reporting/verification process, two were declared not vulnerable by the vendor, six were dropped for being EOL, while one device (DGN2200v2) was declared to *not exist*. Of these, 11 devices that are vulnerable to the exploit (six unsupported and five tested) never received any patches. Only some of the impacted devices are disclosed to the end-user, who remains unaware of any omissions. Worse, attackers can use the public exploit to target the 14 unpatched devices.

3. Properties of the Responsible Disclosure Contract

In this section, we define several key properties that an ideal Responsible Disclosure Contract (RDC) should achieve and discuss the impact of these properties for the end-users. While these properties can be formulated as text, we formalize them here for the sake of a precise definition. Additionally, we use the formal definitions to enumerate potential property violations in Section 6.

Let D be a device, I be a firmware image (that corresponds to a specific firmware version of a device D), and V be a vulnerability found in some image I. Let $\mathcal{VI}$ be the set of all images I (all firmware versions of D) that are vulnerable to V, $\mathcal{AI}$ be the set of all images that vendors have acknowledged as impacted by V, and $\mathcal{PI}$ be the set of all images that have received patches for V.

A device D is vulnerable to V if $\exists I \in \mathcal{VI}$ such that D runs I. A patched device D_p regarding V is a device running a patched image I_p that is not vulnerable to V.

Property 1 (Vendor acknowledgement completeness)**.** *The vendor must acknowledge every vulnerable firmware image.*

$$\forall I \in \mathcal{VI} : \ I \in \mathcal{AI}$$

Violating Property 1 means that the vendor does not acknowledge some vulnerable devices or firmware versions, and, as a result, the end-users running these devices or firmware versions are not informed about the vulnerability status of their devices and cannot apply any mitigations.

As defined, Property 1 applies to all D, even if the device D_{eol} is EOLed, and this leads to the following corollary:

Corollary 1 (EOL device acknowledgement completeness)**.** *The vendor must acknowledge an EOL device D_{eol} as vulnerable to V if D_{eol} is vulnerable to V.*

$$\forall I \in \mathcal{VI} \text{ where } I \text{ runs on } D_{eol} : \ I \in \mathcal{AI}$$

Property 2 (Device patching completeness)**.** *The vendor must provide a patched firmware image for every acknowledged vulnerable firmware image.*

$$\forall I \in \mathcal{AI} : \ I \in \mathcal{PI}$$

End-users must patch their vulnerable devices to mitigate the risk of being exploited. Violating Property 2 means that at least one acknowledged vulnerable device does not receive patched images, and the end-users running these devices will be vulnerable to exploitation. The problem is more severe if the exploit that demonstrates the vulnerability later becomes publicly available as part of the disclosure process (which is the norm), because attackers can simply use the exploit to compromise these unpatched devices.

Property 3 (Device patching correctness)**.** *Every patched firmware image must not be vulnerable to the reported vulnerability.*

$$\forall I \in \mathcal{PI} : \ I \notin \mathcal{VI}$$

Violating Property 3 means that at least one patched device remains vulnerable to the reported vulnerability while the end-users incorrectly believe that their patched devices are secure against the vulnerability.

Property 4 (Patching before disclosure)**.** *The vendor must release patched firmware images before publicly disclosing the vulnerability.*

$$\forall I \in \mathcal{PI} : \ t\_patch(I) < t\_disclosure(V)$$

Violating Property 4 means that the vendor publicly discloses the vulnerability before providing patched images, which allows attackers to exploit unpatched devices.

4. Methodology

We aim to find violations of RDC properties (as Section 3 defines them) in real-world devices. Specifically, we must determine, for a set of devices D and a set of vulnerabilities V, the ground truth of $\mathcal{VI}$, $\mathcal{PI}$, and D_{eol}. This is complicated because (a) the firmware ecosystem

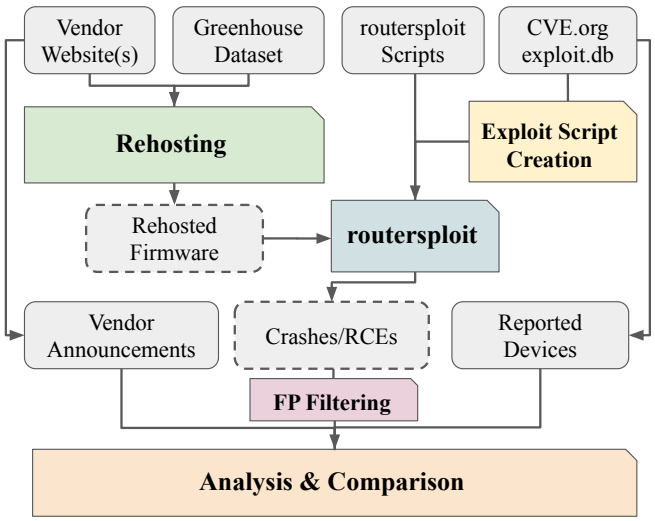

Figure 3: Overview of our analysis methodology, showing how data flows through the four steps of our pipeline from source to analysis.

constantly changes with new images being released, updated, and retired day-by-day [11], and (b) different vendors handle reported vulnerabilities in different manners [39]. We develop a system, BucketLeak, which is a collection of firmware metadata and automated tools for deriving the necessary data to form a ground truth.

4.1. Overview of BucketLeak

BucketLeak starts by creating a large, *empirical* ground truth of vulnerable devices through *running exploits against them*, then matching the empirically obtained data with official announcements from the vendor websites and other third-party sources, such as the CVE database.

We first determine the prevalence of unreported N-Day device vulnerabilities by running exploit scripts of previously reported and/or fixed vulnerabilities against a large set (566) of router and camera devices. Since acquiring hundreds of real-world devices is prohibitively expensive, we rehost the 3,569 firmware images using Greenhouse [52] for each available version of these devices. Figure 3 illustrates our empirical approach to collect data, which can be broken into four major steps:

1) *Device Dataset Creation*, where we gather, label and rehost the devices for our study.
2) *Exploit Dataset Creation*, where we gather CVEs, select and create exploit scripts for replay.
3) *Replay*, where we replay all curated exploits against all curated device firmware.
4) *Verification*, where we conduct manual analysis and verification to eliminate false positives.

With these steps, we generate an empirical ground truth of $\mathcal{VI}$ and $\mathcal{PI}$, from which we can derive DevExPairs. We can then compare this data to the publicly available ground truth of $\mathcal{AI}$, which we source from vendors' security advisories,

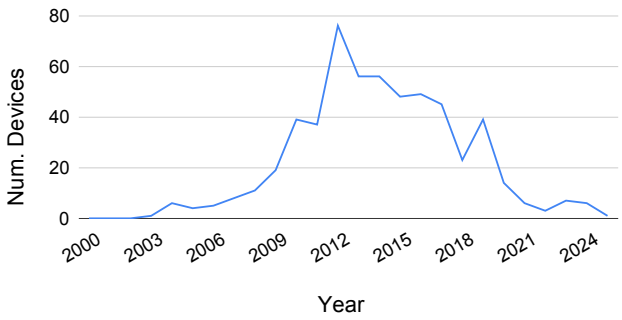

Figure 4: Distribution of firmware images released per year in our dataset.

the CVE database, or other sources (cf. Section 5.1), to identify which exploited devices have or have not been disclosed. Using the metadata of both the device and the exploit affecting it, we map our findings to our Bifurcation Pipeline model, which serves as the basis of our data analysis.

4.2. Empirical Dataset Creation

First, we create our dataset of devices and firmware. We start with the 6,812 images in the Greenhouse dataset [52]. Through using automated web scrapers and manual effort, we extend it to include more devices, arriving at a total of 9,156 images from the same eight vendor brands and four architectures in the original dataset. Figure 4 shows a distribution of collected images over time.

The cut-off date of data gathering was **09/30/2025**, when we completed the manual verification of our dataset for corrupted or missing firmware images. We rehost all images and discard those that failed rehosting. This creates our final Empirical Dataset of 3,569 rehosted images spanning 566 devices.

We also gather the release dates and EOL dates for each device. Some vendors differentiate between EOL and End-of-Service (EOS) devices [5], [21] while others use the terms interchangeably [41]. For the purposes of our study, we consider EOS/EOL interchangeable and conservatively use the later of the two dates as the EOL date.

4.3. Exploit Dataset Creation

Determining if a firmware image is vulnerable to some V is a prerequisite for deriving $\mathcal{VI}$. The most reliable way to determine if an image is vulnerable to some V is to run the exploit against it, so we first must gather a set of exploits.

As of 09/30/2025, routersploit contains 26 exploit scripts targeting one or more vendor brands in our Empirical Dataset. These exploits largely correspond to CVEs from 2010 to 2019. We created 28 additional scripts based on CVEs from 2019 to 2025, bringing the number to a total of 54 exploits. Interested readers may refer to Appendix A for the criteria that we used to select these additional exploits.

We verified each additional exploit script by testing it on at least one reported vulnerable firmware image.

For each exploit script in our dataset, we collect the list of vulnerable devices (and firmware versions) that were *officially disclosed* by the vendor and the list of devices that received a relevant patch. Due to the difficulty of acquiring older firmware images from the respective vendors and the difficulty of rehosting in general, not all devices listed for a given exploit are present in our Empirical Dataset. Thus, we distinguish between devices in our Empirical Dataset and devices that could only be found via external, non-empirical sources such as vendor announcements or third-party write-ups by their associated researcher(s), which we term our External Dataset. These two datasets are used in our model and subsequent analysis.

In total, creating the Empirical, External and Exploit datasets required approximately 12, 5, and 15 human-months of manual effort, respectively.

4.4. Exploit Replay

The final step of our pipeline involves performing large-scale dynamic analysis to empirically identify all firmware images vulnerable to the exploit scripts we found. To this end, we perform an exploit replay of all 54 scripts against all images in our rehosted dataset. To enable post-authentication exploits, we make use of an extended version of the Initializer script used by Greenhouse to perform automated logins and basic setup. Exploit scripts that triggered a potentially valid crash were paired with the firmware image they exploited to form our initial image-exploit set, which we then convert to DevExPairs. For more details on how we detect crashes in our pipeline, refer to Appendix B.

5. Challenges

Building and executing our pipeline involved several challenges that we detail below.

5.1. Determining The Lifecycles of Devices

We list the four main complications that we encountered during the determination of the lifecycle of a device.

Inconsistent device information. Gathering the metadata on device lifecycles (release and EOL dates) is difficult due to diverse publication practices across vendors. Official sources (device vendors) provide lifecycle information in many formats (or none at all), e.g., centralized documents [35], [41], individual webpages for each device, or listing EOL device names in some pages [5], [56]. Unofficial sources are less coherent, with inconsistent technical language, date formats, and labels for different devices. Collating and cross-checking these distinct information sources and mapping them to the devices in our dataset was a key challenge in our study.

Missing release/EOL dates. Some official sources have incomplete information on their own devices' lifecycles.

TABLE 1: Breakdown of image-exploit pairs, DevExPairs, images, and devices at each stage of the filtering process.

	Initial	w/o TPs	Verify Rehost	Verify Trace	Final Set (w/ TPs)
Img.-exploit Pairs	3,708	3,315	2,336	1,798	2,191
DevExPairs	990	883	524	358	467
Firmware Images	1,892	1,834	1,808	1,676	1,760
Firmware Devices	342	320	311	278	312

To handle this, we used multiple sources and prioritized by an approximate order of legitimacy. We first checked official sources (e.g., vendor's website) for a device. For cases where official sources were unavailable, we searched a number of reputable third-party websites, e.g., FCC reports [26], community-maintained wikis [53], and listings by larger resellers [3], [24]. If unsuccessful, we looked for announcements, reviews, and technical reports that mention the device to estimate its release and/or EOL dates. As a last resort, we used the dates of the first and last firmware images in our dataset. Overall, we perform a conservative estimate of release and EOL dates corresponding to the widest range of time for each vendor.

Measuring active devices. The number of active devices on the web is difficult to measure. Not all real-world devices are connected 24/7, while others might have firewalls or other security features that block attempts to identify the type of the device, which makes retrieving information on these devices inconsistent. To estimate the number of devices in circulation, we based our approach on the Aliveness Analysis [59]. We ran weekly queries on ZoomEye [62] over four weeks from April to May 2025 to find the average number of active instances for each device in our dataset.

Regional variants. Several vendors maintain regional variants of devices, each with their own lifecycle. For example, D-Link DIR-825 has US, EU, and IN versions. The EOL date of DIR-825 US was 09/01/2015, while the UK version was considered EOS on 01/01/2016. Meanwhile, the IN version is still considered live and supported. In these cases, we consider a multi-regional device to be still supported (non-EOL) as long as one variant is officially supported anywhere at the time of dataset completion, 09/30/2025.

5.2. Dataset Verification

Our initial replay of 54 exploits against 3,569 rehosted firmware images generated 3,708 potentially vulnerable image-exploit pairs (corresponding to 990 potential DevExPairs). We assume that any image-exploit pair that successfully performed a remote-control execution (detected via the exploit script itself, or via the presence of a unique hashfile generated via a *touch* command) to be a true positive. This still leaves a large set of 3,315 potentially invalid exploits; to filter them, we need to consider two types of false positives caused by our methodology, (1) false, "erroneous" crashes caused by incomplete rehosting, and (2) actual crashes in the

binary that were caused by program behavior unrelated to the nature of the vulnerability targeted by our exploit script.

Table 1 summarizes the progression of our image-exploit pairs and DevExPairs through the filtering process. We now describe steps in more detail.

Verification of rehosting. From our initial set of image-exploit pairs, we ignore the 393 successful RCE cases to focus on the remaining 3,315 image-exploit pairs that need further verification. Due to limited time and resources, we could not manually verify all 1,892 corresponding firmware images for incomplete rehosting. We instead select the top 50 images (2.5%) with the highest number of image-exploit pairs, based on the intuition that a poorly rehosted image would be more unstable and thus subject to more crashes. We verify the rehosting status of these firmware images through manually interacting with their rehosted webserver and examining the traces leading up to the generated crashes, a process that took approximately 40 human-hours to complete. From this set, we find 27 images to be incorrectly rehosted and unusable for any sort of analysis. Removing these leaves us with 1,865 potential true positives.

Trace-based verification. To verify if the remaining crashes are valid, we make use of a combination of both automated and manual analysis. First, we recognize that targets that share similar vulnerabilities will run through similar blocks of code before a crash. We use a Python script to compare the stack and file operations at the point of crashing for both the target and a known vulnerable image (disclosed via CVE or vendor announcement) that we previously validated to be a true positive. Cases which show a large divergence ($<$0.9 weighted similarity score comparing stack and file traces) are considered to be false positives, as they are likely crashes unrelated to the exploit reported. The subsequent case study illustrates the logic behind how our automated script filters out false positives using two crashing targets, Netgear devices R8000_V1.0.3.32_1.1.21 and wnr612v3_wnr500_V1.0.0.18, for the exploit script derived from CVE-2024-12147.

Case Study: Confirming a true positive. CVE-2024-12147 lists the Netgear R6900_1.0.1.26_1.0.20 as an affected device, one that BucketLeak has successfully rehosted and exploited. The execution trace indicates that it crashes inside a specific function, `sub_1B814`. This function invokes `memcpy()` on a buffer with unchecked lengths, exactly as described in the CVE report. The execution trace of another device, R8000_V1.0.3.32_1.1.21, also maps to a function with similar behavior. Furthermore, the strace prior to the crash shows a similar pattern of NVRAM accesses.

By examining the stack trace at time of the crash, we see that the stack in both the reference and target execution have a similar pattern of bytes, as both program stack frames are overflown in the same manner. We also manually confirm that both R8000_V1.0.3.32_1.1.21 and R6900_1.0.1.26_1.0.20 exhibit similar crashing behavior to our exploit by mapping their execution trace to disassembly code. As Figure 5 shows, the functions up to (and including) the crash location are almost identical. Thus, by comparing the file operations (*open, close*) and network operations (*recv, accept*) prior to the crash for both the reference and target firmware images, we confirmed that two programs have similar data flow behavior leading up to the crash, reflecting a similar pattern of memory corruption.

Case Study: Filtering out a false positive. The syscall trace of wnr612v3_wnr500_V1.0.0.18, on the other hand, shows a different pattern of network and file I/O operations from the syscall trace of the reference target. The resulting stack memory when the crash occurs also shows a different byte pattern. By mapping the execution trace of wnr612v3_wnr500_V1.0.0.18 to its decompilation, we locate the crashing function `sub_40B528`, which does not resemble that of our reference image (Figure 6). The crash occurs at a call to `free()` on an unchecked NULL pointer and is not the stack buffer overflow we have seen before. Although the crash is a valid vulnerability, we consider it a different vulnerability from the one targeted by CVE-2024-12147 and, thus, a false positive.

In cases where we do not have a reported, rehosted firmware image that can be used to validate our exploit, we perform the comparison between all potentially affected images and cluster those with similar traces together. We then manually verify one sample from each cluster to find one that fits the behavior of our reported vulnerability, discarding the entire cluster if it does not.

We filter out 606 false positive image-exploit pairs and combine the resulting set with the 393 true positives, obtaining 2,191 image-exploit pairs. These correspond to 467 DevExPairs, of which 260 are confirmed to not be patched in their latest version nor disclosed to the public. Our pipeline thus provides a quantifiable lower-bound estimate on the number of devices inside the "invisible gap."

6. Data Analysis

In this section, we discuss the results of our large-scale analysis. Figure 7 presents the full results, which follows the same design as the pipeline specific to our exemplary case study, CVE-2017-6334 (cf. Figure 1). The first section, *Dataset Breakdown*, groups the 599 DevExPairs in our Empirical and External-Sources-Only datasets based on whether we managed to empirically run the exploits against this device. The second section, *Observed Response*, shows how these DevExPairs map from the source(s) to the resulting actions taken by their respective vendors as observed in vendor announcements. For the majority of DevExPairs with no external sources (i.e., undisclosed N-days in devices), we conduct an *Additional Empirical Validation*. This uses our knowledge of empirical exploitability and the devices' lifecycles to determine the number of DevExPairs for which no patch exists. First, we apply a simple heuristic: If we can exploit the latest firmware version for that device, then we consider all firmware versions of that device as vulnerable.

If the latest version is not exploitable (or not rehostable), we take a more cautious approach to evaluating if DevExPairs are true positives. When we could not locate or

```
int __fastcall sub_1A480(unsigned __int8 *src)
{
    int v2; // r8
    ...
    const char *v13; // r0
    char s[140]; // [sp+4h] [bp-8Ch] BYREF

    memcpy(&unk_EFADD0, src, 0x31u);
    byte_EFAE02 = 0;
    if (strcmp((const char *)src, "*#$^"))
        return -1;
    v2 = src[39];
    ...
    src[37] = 0;
    src[36] = 0;
    src[38] = 0;
    v9 = v5 + (v6 << 8) + (v7 << 16);
    src[39] = 0;
    memset(s, 0, 0x64u);
    memcpy(s, src, v9);
    calculate_checksum (0, 0, 0);
    calculatechecksum (1, s, v9);
}
```

```
int __fastcall sub_1B814(unsigned __int8 *a1)
{
    int v2; // r8
    ...
    const char *v13; // r0
    char s[140]; // [sp+4h] [bp-8Ch] BYREF

    memcpy(&unk_F48078, a1, 0x31u);
    byte_F480AA = 0;
    if (strcmp((const char *)a1, "*#$^"))
        return -1;
    v2 = a1[39];
    ...
    a1[37] = 0;
    a1[36] = 0;
    a1[38] = 0;
    v9 = v5 + (v6 << 8) + (v7 << 16);
    a1[39] = 0;
    memset(s, 0, 0x64u);
    memcpy(s, a1, v9);
    calculate_checksum (0, 0, 0);
    calculate_checksum (1, s, v9);
}
```

Figure 5: Comparison of decompilation between reference firmware *R6900 1.0.1.26_1.0.20* and *R8000_V1.0.3.32_1.1.21*. The highlighted portion indicates the code block where the crash was detected.

```
  ...
    v19 = snprintf(
      v14,
      500,
      "<HR>\n<ADDRESS><A
↪   HREF=\"%s\">%s</A></ADDRESS>\n<BODY>\n <HTML>\n",
      "http://www/acme.com/software/mini_httpd/",
      "mini_httpd/1.19 19dec2003");
    sub_40647C(v14, v19);
    free(v14);
    sub_403954();
    if ( !a3 )
    {
LABEL_19:
      close(child_accept_fd);
      sub_402D90();
      exit(1);
    }
    v10 = a3;
LABEL_18:
    free(v10);
    goto LABEL_19;
    }
  }
  syslog(2, "out of memory exiting");
  goto LABEL_19;
```

Figure 6: Decompilation of *wnr612v3_wnr500_V1.0.0.18*, highlighting the free() on an unchecked NULL pointer.

rehost the latest firmware image, we consider the DevExPairs unverifiable. The now remaining DevExPairs are those for which we could exploit an older version but not the latest rehosted firmware image for that device. This could mean that the device is patched in its latest version or that the vulnerability was masked by small differences in the dynamic analysis, either caused by the rehosting process or small changes in the UI that differentiate it enough to seem immune.

Because we could not empirically validate the 70 DevExPairs extracted from CVE/GitHub sources that we lacked a rehosted image for, we do not count them towards the N-days confirmed by our pipeline.

TABLE 2: Breakdown of vulnerable, reported, and unreported devices for each brand.

	Rehosted	Vulnerable	Reported	Unreported
Asus	161	110	58	110
Belkin	25	21	2	21
D-Link	96	62	54	44
Linksys	17	8	14	8
Netgear	141	90	34	86
tenda	30	8	3	8
TPLink	34	0	3	0
Trendnet	62	13	2	13
TOTAL	566	312	170	290

6.1. A Bird's-Eye View: Reporting to Patching

Table 2 breaks down the number of rehosted devices in our dataset by vendor, along with the numbers of devices with reported and unreported vulnerabilities for each vendor. By breaking down our discovered crashes into DevExPairs, we reduce the initial number of 2,191 crashes into 422 DevExPairs, which are undisclosed N-days. We further reduce them to 260 previously unknown DevExPairs that are not patched in any firmware versions, i.e., affect the latest versions of the vulnerable devices. When using ZoomEye to estimate the active number of these still-vulnerable devices, we find 1.04 million (95% confidence interval, or CI, for a range of 0.84-1.24 million) potential targets. A malicious actor can attack these devices even they are upgraded to the latest firmware.

The vendors' awareness of a DevExPair does not guarantee a patch. The responsible disclosure process is meant to mitigate this by providing guidelines to limit the potential risk of publicly sharing vulnerability knowledge instead of attempting to hide it via obscurity. The observed response in Figure 7 however, shows that the *majority* of DevExPairs sources do not make it through the pipeline to be patched.

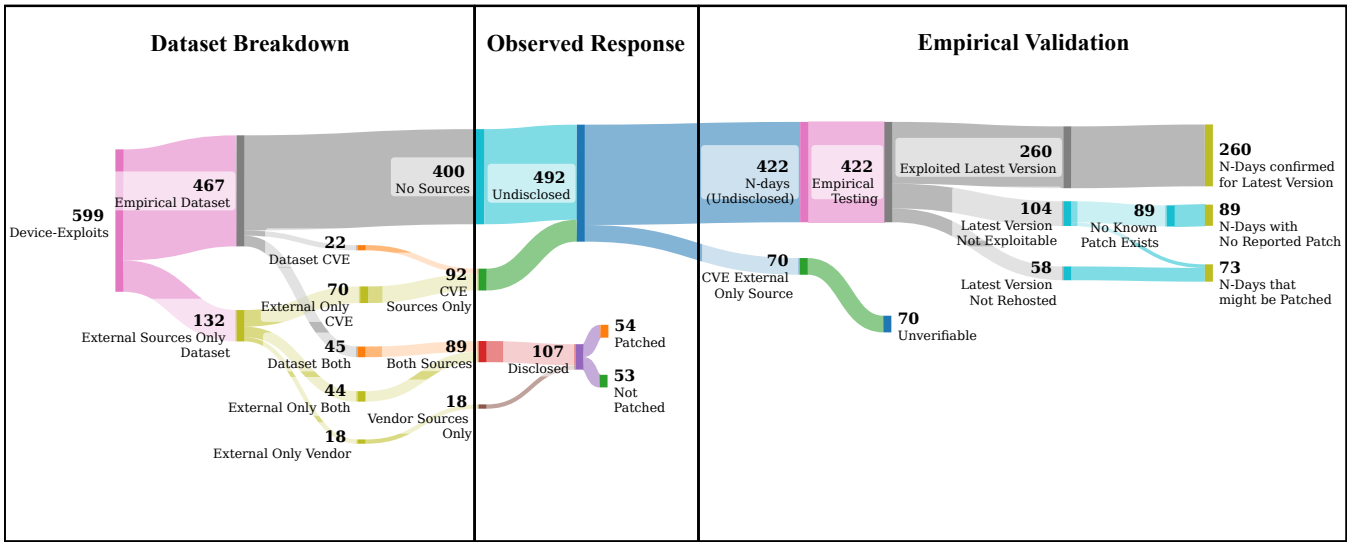

Figure 7: Bifurcation Pipeline showing the breakdown of DevExPairs in our combined dataset by sources, whether they were disclosed, and if they received a patch based on vendor announcements or our own empirical validation.

46.2% (92/199) of unique DevExPairs with at least one CVE information source on the internet were never disclosed by their vendor. Of these undisclosed DevExPairs, 22/92 (23.9%) were found to be exploitable by our pipeline. In contrast, the set of 107 disclosed DevExPairs comprise a mere 17.9% of the 599 in our total combined dataset. These correspond to approximately 250,000 active devices (95% CI, 202,000-298,000) found via ZoomEye, an order of magnitude fewer affected real-world devices compared to undisclosed DevExPairs.

Of the remaining 492 undisclosed DevExPairs we managed to exploit via our pipeline, 92 (15.4% of our total 599) had corresponding CVE descriptions and/or scripts that mentioned the specific DevExPairs but are not listed on their vendors websites. While a security manager might know to check NVD or websites outside that of the vendors, the average end-user does not. Even then, there are 422 DevExPairs that were previously undisclosed until tested with our pipeline, corresponding to 1.71 million devices (95% CI, 1.37-2.04 million) still in circulation, of which 1.04 million (95% CI, 0.84-1.24 million) are still vulnerable in their latest firmware version. This represents a ratio of *four* times the number of devices with undisclosed vulnerabilities compared to disclosed ones (between 2.8-6.1 times at 95% CI).

6.2. P1 Violations: Incomplete Acknowledgement

Violations to Property 1 can occur in two ways: (a) Some firmware image I is vulnerable to some vulnerability V but not included in the disclosure ($\exists I: \ I \in \mathcal{VI} \wedge I \notin \mathcal{AI}$), and (b) some I is in the disclosure but is not vulnerable to V ($\exists I: \ I \notin \mathcal{VI} \wedge I \in \mathcal{AI}$). The former case is more common, although our study found a comparable number of the latter.

Ideally, coordination between researchers and vendors should lead to consistency between vulnerabilities disclosed by the vendor and those disclosed independently by the researcher. Of the 199 DevExPairs that we found an external source for, only 107 (54.8%, slightly more than half) DevExPairs were also included in a vendor announcement. An approximately equal (46.2%) number of DevExPairs out of 199 had only CVE entries or write-ups, of which 22 we could empirically verify. While some of this is attributable to the limitations of our pipeline, better rehosting and sourcing would only increase the number of CVE sources that can be targeted. This ten percent of empirically tested, publicly known device-exploit pairs (DevExPairs) thus provides a conservative approximation of the number of DevExPairs for which publicly available scripts are easily available for testing yet not mentioned in any vendor source.

Observation 1. The quality and quantity of information sources is highly inconsistent and sometimes inaccurate.

Case Study: Unacknowledged vulnerabilities. CVE-2025-6121 was disclosed on NVD on 06/16/2025. It lists the D-Link device DIR-632 as vulnerable to a stack-based buffer overflow, with an accompanying GitHub link containing a detailed explanation by the researcher and an accompanying PoC. However, as of 09/30/2025 three months later, no vendor announcement or patch about this vulnerability can be found on the D-Link website. We were able to confirm via our pipeline that the latest firmware version of DIR-632 remains vulnerable to this exploit. This vulnerability affects over 150,000 devices that we found via ZoomEye belonging to various users, many which are not aware that their device has a vulnerability with a working exploit publicly available online for more than last three months.

The disclosure process is ultimately a means for the sharing of vulnerability information between stakeholders of an affected device. When multiple parties each work with partial, sometimes conflicting information, it can further mask any actual issues that are overlooked. This in turn

exacerbates the false sense of security that arises despite the significant ratio of undisclosed DevExPairs to disclosed ones, as discussed below.

Observation 2. Devices listed in the disclosure process represent only a fraction of all at-risk targets.

Supported $\neq$ Secure. Besides newer vulnerabilities affecting EOL devices, older vulnerabilities that were only reported for these EOL devices may potentially affect newer, still supported models. Code reuse is common in the industry, especially given the number of similar functions the average router, camera or switch has. This includes any vulnerabilities that might have been discovered but not patched in the firmware image for these older devices that then get reused in a newer one.

Of the 41 exploits that affect at least one EOL target, 13 also affect one or more devices that are still supported as of 09/30/2025. Of these 13 exploits, four were disclosed for an EOL device, but not for the still supported device. The following case study provides one such example, showing how ignoring reports for EOL devices can lead to future security risks.

Case Study: CVE-2024-26342. The Asus devices RT-AC68U and RT-AC58U from our dataset share the same vulnerability to CVE-2024-26342, disclosed to the public on 2/28/2024 [45]. The Asus RT-AC68U reached EOL on 3/20/2017 [5], while RT-AC58U had yet to reach End-of-Life at the time, but was not included in the report. CVE-2024-26342 was reported for RT-AC68U on 3/20/2017 [45] but was, to the best of our knowledge, neither patched nor disclosed on the Asus website. Yet the RT-AC58U, which still received firmware updates up to 12/05/2024, was found to retain the same vulnerability by our pipeline.

As seen from the decompilation of the vulnerable sections in Figure 8, the two firmware binaries share very similar code. A similarity metric using BinDiff on the webserver binaries of the two devices shows that 60% of the functions (including the vulnerable one targeted by the exploit) share over 95% of their code, suggesting a high degree of code reuse between the two devices. Thus, a patch for RT-AC68U could likely apply to RT-AC58U, reducing the window of vulnerability for the newer device with minimal additional effort. EOL devices for reported vulnerabilities represents an unnecessary and significant increase in the security risk of future devices, as they are released with a vulnerability to older, known exploits.

6.3. P2 Violations: Incomplete Patching

The violation of Property 2 occurs when some firmware image I is vulnerable to some V but not patched ($\exists I : I \in \mathcal{VI} \wedge I \notin \mathcal{PI}$). A key reason for P2 violations is the reluctance of vendors to patch EOL devices.

Figure 9 breaks down the vendor announced reasons for not patching disclosed DevExPairs. We see that the majority (39/53) of disclosed-yet-unpatched DevExPairs are for unsupported devices that have reached End-of-Life.

TABLE 3: Number of Exploits, EOL devices, pre-EoL, and post-EOL DevExPairs organized by Vendor.

	# Exploits	# EOL Devices	# Pre-EOL DEXs	# Post-EOL DEXs
Asus	5	99	27	111
Belkin	2	21	0	45
D-Link	17	56	58	43
Linksys	8	6	3	5
Netgear	14	81	38	105
tenda	4	3	7	1
TPLink	2	0	0	0
Trendnet	2	13	6	18
TOTAL	54	279	139	328

Meanwhile, the number of patched devices more heavily favors non-EOL devices (47/54).

At first, this appears to be a reasonable prioritization of developer effort, focusing on newer devices that will continue to see increasing usage while older models are phased out. Most vendors thus encourage users to replace the expired device with a newer one that is still supported. In an ideal scenario, all devices that are no longer supported by their vendor are immediately pulled from circulation. However, in practice, many consumers continue using the same EOL device for years after they are no longer supported [4], [59]. While continuing to support an ever-increasing number of older device models is not feasible for anyone, we argue that EOL devices are being under-prioritized relative to their potential vulnerability impact. Removing devices from consideration for support requires a more nuanced strategy than a simple cut-off date.

Observation 3. End-of-Life devices that usually go unpatched have some of the highest vulnerability impact.

6.4. Impact of EOL Devices

Past research [4], [59] shows that the majority of discoverable, active firmware devices are currently EOL. Using ZoomEye for the devices in our dataset, we find approximately 60,800 vulnerable non-EOL devices corresponding to 15 DevExPairs in our dataset that are active and connected to the Internet. This is in contrast to the 1.88 million physical devices that are EOL which map to 279 DevExPairs, more than 30 times the number of their non-EOL counterparts.

These End-of-Life DevExPairs are not limited to vulnerabilities found during the device's original lifecycle. Table 3 shows the total number of EOL devices in our dataset divided by brand, the number of post-EOL DevExPairs found their EOL date, and the number that were eventually patched. As our pipeline can only confirm if a device is unpatched in its latest version, we take a conservative approach and consider unconfirmed, potentially exploitable DevExPairs to be patched in their latest firmware version. Overall, 328 DevExPairs were discovered *after* their EOL date, in contrast to the 139 DevExPairs that were discovered during its operational lifecycle. These 328 DevExPairs correspond to 1.65 million active devices found on ZoomEye, over 94.1% of the total number of devices with at least one undisclosed vulnerability.

```
int __fastcall sub_DAC8(const char *a1, int a2, int a3){
    ...
    if (strchr(a1, 13))
        v4 = strtok(v4, "\r")
    v6 = strstr(v4, "//");
    if (v6 && (v4 = v6 + 2, strchr(v6 + 2, '/'))) {
        v7 = strtok(v4, "/");
        v8 = a2;
    }
    ...
    strlcpy(v8, v7, a3);
    return a2;
}
```

```
int __fastcall sub_D894(const char *a1, int a2, int a3){
    ...
    if (strchr(a1, 13))
        v4 = strtok(v4, "\r")
    v6 = strstr(v4, "//");
    if (v6 && (v4 = v6 + 2, strchr(v6 + 2, '/'))) {
        v7 = strtok(v4, "/");
        v8 = a2;
    }
    ...
    strlcpy(v8, v7, a3);
    return a2;
}
```

Figure 8: Decompilation of the vulnerable section for RT-AC58U and RT-AC68U, showing near-identical code.

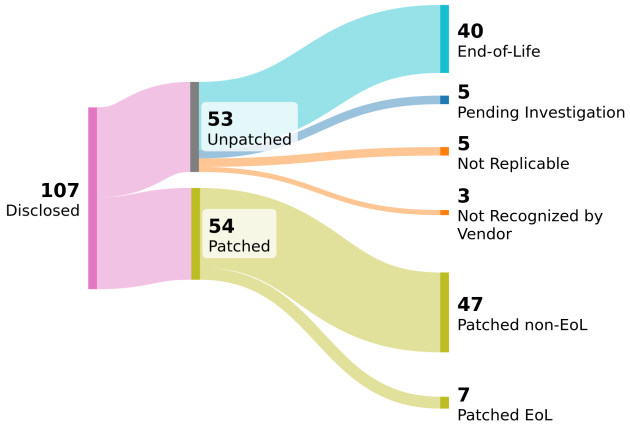

Figure 9: Breakdown of unpatched DevExPairs in the disclosed set by (announced) reason.

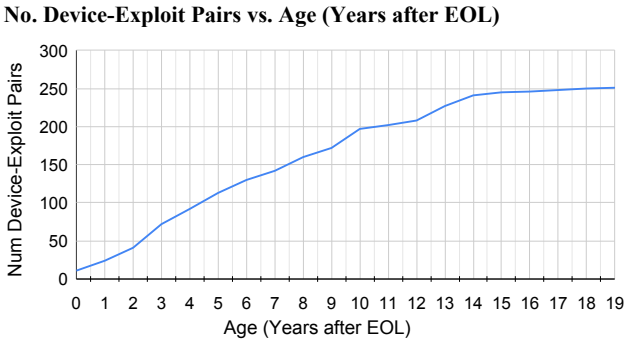

Figure 10: Line graph showing number of newly discovered DevExPairs against the number of years since its affected device has reached EOL.

The portion of affected EOL devices in our dataset might suggest that the issue is with the EOL policy the vendors hold rather than the disclosure process. However, even if we ignore EOL devices, the overlooked gap is still significant: Of all 566 unique devices in our dataset, 312 (55.1%) are vulnerable to at least one exploit in our pipeline. Of these 312 device models, 139 (44.6%) were non-EOL at the time the exploit was released to the public. These 139 devices correspond to 1.16 million active devices (95% CI, 1.33-1.97 million) discovered via ZoomEye. Despite the exploits becoming public knowledge, 109 of these 139 devices remain undisclosed and unpatched to this day, corresponding to 0.85 million active devices (95% CI, 0.69-1.01 million), over 73.1% of the number of active real-world devices that still have an undisclosed, unpatched exploit that had been made known to the public while they were still non-EOL. So, while EOL policies contribute to the issues observed in this paper, they are not the root cause.

Our results show how EOL policies amongst vendors are a detriment to the responsible disclosure process in their current form. Immediately cutting support for EOL devices causes a significant number of vulnerable devices in circulation with vulnerabilities that are overlooked (or ignored) by design. This is exacerbated by the current policy of releasing exploits publicly through the responsible disclosure process regardless of whether a model is EOL or the vendors' design to disclose or patch a device. Note that we do not argue to limit publicly released information; rather, we believe vendors should revisit their EOL policies to ensure lasting protection of their customers.

Figure 10 shows the rate of increase for these post-EOL DevExPairs against the number of years that have passed since. The rate of discovery for DevExPairs eventually approaches a limit, but only starts slowing down around the 3-4 year mark. While it is expected that the number of exploits affecting a device increases over time until it stops receiving support, our data supports the idea that gradual deprecation (through a hybrid EOS/EOL plan some vendors are implementing) is significantly more secure compared to a single cut-off date. Furthermore, greater measures need to be taken to encourage consumers to phase out older EOL devices in their possession for newer models, or to better secure their outdated firmware from being easily found over the public Internet.

6.5. P3 Violations: Incorrect Patching

Incorrect patching is rather rare, but we spot an instance in our dataset that violates Property 3 that we describe below.

Case Study: Nonexistent patches. D-Link device DCS-5009L is listed on both the NVD entry for CVE-2019-10999

and referenced in the D-Link announcement SAP10131 as vulnerable for version 1.08.11 and lower. The announcement specifically mentions a patch for the DCS-5009L, v1.1001, released in July 2019 [20]. However, when searching for firmware, we found no such version of DCS-5009L, not even on the D-Link website. The latest version available was for v1.09.12, which is a version higher than the reported firmware. While we would normally attribute this to a labeling error in the SAP announcement, we could exploit DCS-5009L 1.09.12 using the exploit for CVE-2019-10999. This suggests two possibilities: Either a patched version does exist, but was never released. Or a patched version does not exist, and the actual latest firmware version was mistakenly reported as patched. Either way, this single case demonstrates a real-world violation of P3, as a device reported as patched in its latest version does not have any such firmware available.

6.6. P4 Violations: Late Patches

The violation of Property 4 occurs when some firmware image I is vulnerable to some V and is patched only after public disclosure of V ($\exists I : I \in \mathcal{VI} \wedge I \in \mathcal{PI} \wedge \text{t\_patch}(I) > \text{t\_disclosure}(I)$).

Using data collected from public sources on the dates of announced and patched devices, we find that 33/170 (19.4%) of devices reported for 54 exploits were patched *after* the exploit was disclosed, or not patched at all.

6.7. Vulnerability Over-reporting

The categorization and grouping of vulnerability information is another significant inconsistency in the disclosure process. When building our Exploit Dataset, we found that eight of the 54 exploits in our dataset were overreported. These exploits have the same base vulnerability, but different device targets, usually similarly named devices belonging to the same brand. For example, CVE-2024-22651 and CVE-2022-3561 are both remote-command-execution vulnerabilities that target the same function, `ssdpcgi_main`, in two different firmware devices, DIR-818LW and DIR-815 [43], [44] and yet were reported more than two years apart. Disclosures in both vendor and CVE sources sometimes list multiple devices per disclosure, or just a single device per announcement. They may count multiple firmware-exploits for the same device as separate, or lump them all together. Overreporting then bloats the amount of information disclosed, potentially leading to redundant work and wasted effort by other security researchers in the future.

7. Limitations

Our methodology has several limitations that we were unable to fully address.

False Positives. Even with our filtering steps, there are still false positives in the final list of 599 DevExPairs. As Section 5.2 discusses, these false positives either arise from differences between the rehosting environment and the actual device, or are legitimate but unrelated crashes caused by the exploit. We evaluated the impact of these false positives by randomly sampling 60/422 DevExPairs and manually confirming that the crash triggered reflects the nature of the exploit replayed against it. 60 samples ensure 90% confidence with 10% margin of error in our results. The analysis process of the false positives took approximately 50 human-hours to complete.

We found that 47/60 (81.7%) of these sampled DevExPairs are true positives. Four of the 13 false positives are DevExPairs that could not be reproduced, either due to rehosting differences (i.e., the exploit triggered a crash due to a missing NVRAM value in the rehosted environment causing a segmentation fault), or because the exploit targeted a specific range of memory values (such as a heap overflow that assumes characteristics of the heap allocator, which differs between the emulated and real environment). The remaining nine false positives were crashes in unrelated parts of the code. Although these DevExPairs were not what was targeted by the replayed exploit script, they are still potential vectors for attacks.

Notably, for the nine false positives that arose from an unrelated crash, four DevExPairs were buffer overflow vulnerabilities that could lead to hijacking the program counter. Even though they are not the intended vulnerability, they are still exploits that generally belong in the same CVSS Severity Category ("High") as the original CVE. The remaining five unrelated DevExPairs are NULL pointer dereferences that led to a crash. These generally belong to a lower Severity Category ("Medium") but can still lead to denial-of-service attacks. None of these unintended exploits were previously reported for those devices.

Despite a false positive rate of 18.3% after filtering, 9/13 false positives still represent valid 0-days that affect devices in our dataset. Feeding these newly discovered crashes back into the BucketLeak pipeline as additional exploits to replay could potentially increase the number of discovered DevExPairs even further. Future work could look into incorporating this feedback mechanism into the BucketLeak pipeline to gain a better understanding of how exploits of a similar nature are handled in the disclosure pipeline and their subsequent effect on the perceived vulnerability of a given device.

Rehosting Limitations. Not all firmware versions we gathered for our dataset were rehostable. As we depend on a rehosting tool to perform large-scale emulation of our targets for testing, we also inherit the limitations of the rehosting platform. Our choice to use Greenhouse was based on ease of instrumentation and the relatively large rehostable dataset it provided. It is likely that combining Greenhouse with other full-system rehosting platforms would significantly expand our testing dataset and help reduce the number of false positives.

Dataset Representation. The firmware devices in the dataset used for our evaluation is based on the ones used by past large-scale firmware analysis papers such as Greenhouse and FirmAE. These datasets focus on firmware from

Linux-based router and camera devices, which form only part of the myriad of devices in the Internet-of-Things. Other types of firmware, such as monolithic firmware are not considered in our study. Searching for the keywords "firmware" and "router" on CVE.org shows approximately 7,700 results for known firmware exploits and approximately 2,400 results for router based exploits. Focusing on router devices thus remains representative of IoT devices for the purposes of our vulnerability impact study.

High Manual Effort. The majority of human effort in this study was spent on annotating and verifying the data we found on the lifecycle of each device. Many exploits in our dataset also had to be crafted manually from human-readable descriptions or limited information by the original discoverer. As such, we only managed to test our dataset on 65 different exploits, most of which are more than four years old, as these were the scripts that were already written as part of the routersploit platform. While we already tried to offset it with a few self-created exploits drawing from the last two years of CVEs, future work could look to expand this list further with a much larger number of recent exploits targeting newer devices.

Unobservable Real-world Devices. Finding information on active devices is difficult due to the opacity of devices on the public internet, as well as third-party reseller sites like eBay and Amazon. While ZoomEye provides a decent estimate, it is likely a vast understatement as the majority of home devices would be behind a NAT or not connected altogether. Processing the circulation of second-hand embedded devices on the market is in itself a significant undertaking, especially given how much the numbers may shift over time. Our study attempt to use the numbers gathered as a conservative estimate of the risks posed to real-world devices, as the actual number is likely to be significantly higher.

8. Related Work

In this section, we discuss prior work on responsible vulnerability disclosure, focusing on its challenges, influencing factors, and domain-specific perspectives in IoT, embedded systems, and beyond.

Research on responsible vulnerability disclosure processes has examined the critical factors that influence how vulnerabilities are reported and addressed. Cavusoglu et al. [16] identified key challenges when a vulnerability affects multiple vendors, showing that optimal disclosure policies cannot always ensure timely patching and that incentives for early discovery and early warning systems influence patch release timelines. In a follow-up study [17], they also explored the efficiency of disclosure mechanisms from a knowledge management perspective, finding that responsible disclosure policies generally encourage patching but that vendor incentives, vulnerability characteristics, and cost structures critically shape outcomes. More recently, Bai et al. [8] revisited the effectiveness of disclosure, highlighting delays and ignored reports even for security-critical vulnerabilities, and pointing to report quality and author experience as factors in patching timelines. Collectively, these works emphasize that while disclosure processes are essential, their efficiency and consistency remain limited.

Within the context of IoT and embedded systems, prior work has noted unique challenges in extending responsible disclosure to device ecosystems. Wang et al. [59] conducted the first large-scale measurement study on EOL devices, revealing that millions of active EOL devices remain vulnerable due to discontinued vendor support, with many threatened by high-risk vulnerabilities. Ding et al. [23] studied bug bounty programs and responsible disclosure as mechanisms to improve IoT security management, suggesting that crowd-sourced ethical hacking can enhance the vulnerability reporting and remediation practices.

Other domains have also been examined to better understand the vulnerability disclosure dynamics. Håvanå [29] studied the communication processes in software vulnerability reporting, revealing that one-way communication between reporters and vendors often hinders effective vulnerability management. They highlighted the importance of trust, procedural knowledge, and codification. Similarly, Böhme et al. [10] analyzed responsible disclosure in the cryptocurrency ecosystem, underscoring unique challenges such as decentralized coordination, financial incentives, and the high stakes of delayed fixes in adversarial environments.

While prior research has explored disclosure mechanisms, IoT lifecycle vulnerabilities, and communication challenges, little attention has been paid to the "invisible gap" where both unsupported EOL devices and overlooked supported devices remain vulnerable despite being reported. By quantifying this gap and analyzing real-world case studies, our work sheds light on an often-missed source of ongoing security risk in embedded systems.

9. Conclusion

With the growing number of IoT devices today, keeping track of all the security vulnerabilities and risks involved can be overwhelming for the average consumer. While systems do exist to manage this information, they are limited by what is prioritized and disclosed in the process. Assuming that no news is good news, that an unreported device is naturally safe, leads to otherwise addressable vulnerability risks propagating through the firmware ecosystem.

In this paper, we examine the potential vulnerability impact that undisclosed devices exploits have on the security of active devices in the real-world. We highlight through various case studies and examples the ways in which code reuse and incomplete information can lead to significant swathes of vulnerable targets going unnoticed, creating a growing gap in the public consciousness of the risk these devices pose. While we are unable to give a complete picture of the entire gap, what we do manage to quantify forms a worrying underestimate of the sheer number of unnoticed, high-risk devices that are still in circulation. It is our hope that this study helps draw attention to these issues and encourage future research into the matter.

10. Acknowledgement

The authors would like to thank the shepherd and anonymous reviewers for their guidance and feedback, as well as the generous support of the US Department of Defense. This project has received funding from the following contracts: Air Force Research Laboratory (AFRL) FA9550-24-1-0204; Advanced Research Projects Agency for Health (ARPA-H) 1AY2AX000167-01; Defense Advanced Research Projects Agency (DARPA) N6600122C4026 and HR001118C0060; National Science Foundation (NSF) 2232915 and 2146568; Office of Naval Research (ONR) N00014-23-1-2563.

Ethics Considerations

Our work does not include any studies that make use of any human subjects. We obtain our data entirely from publicly available digital sources and our own work. All testing of exploits and vulnerabilities are conducted on emulated targets in our own internal network. No actual third-party devices were attacked or exploited in the process.

Our dataset is drawn from publicly available data, such as published exploits, known CVEs, and firmware available on the official vendor websites. Consideration is given to how much older firmware has been removed from the websites after we scraped it. While we understand this might reflect a desire by vendors to emphasize the lack of support for these firmware images, we believe that security via obfuscation is ineffective, and it is more important that these images receive proper scrutiny to benefit the embedded community at large. Furthermore, many of these images are also available on third-party websites such as Softpedia[3], which we believe minimizes the impact of their inclusion in our dataset.

Our study draws attention to the large number of unreported vulnerable devices and their corresponding exploits. It is possible for a malicious actor reading our paper to make use of this information to target the aforementioned devices. We mitigate this through two considerations. Firstly, similar to the thought process behind disclosing vulnerabilities even if a patch is not available, given that the exploits are already public or based on public information, malicious actors could already be employing them in private. Therefore, discussing them here would benefit the public by making this information available. Secondly, we are in the process of responsibly disclosing the unpatched non-EOL devices that we found to be vulnerable to the vendors. As there are many, this process is ongoing.

References

[1] ACM CCS. Call for Papers. https://www.sigsac.org/ccs/CCS2025/call-for-papers/, 2025.

[2] Kendra Albert. Everything Old Is New Again: Legal Restrictions on Vulnerability Disclosure on Bug Bounty Platforms. https://www.usenix.org/conference/usenixsecurity25/presentation/albert, 2025.

[3] Amazon.com. https://www.amazon.com/s?k=router. (Accessed on 2025-10-30).

[4] Manos Antonakakis, Tim April, Michael Bailey, Matt 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. Understanding the Mirai Botnet. In *USENIX Security Symposium*, 2017.

[5] ASUS. ASUS End-of-Life Product List. https://www.asus.com/event/network/eol-product/. (Accessed on 2025-10-23).

[6] ASUS. ASUS Product Security Advisory. https://www.asus.com/security-advisory/. (Accessed on 2025-10-23).

3. Softpedia (https://www.softpedia.com/) is a software and tech news website that indexes, reviews and hosts downloadable software and reports news on technology and science topics.

[7] Jessy Ayala, Steven Ngo, and Joshua Garcia. A Deep Dive Into How Open-Source Project Maintainers Review and Resolve Bug Bounty Reports. In *IEEE Symposium on Security and Privacy (S&P)*, 2025.

[8] Weiheng Bai and Qiushi Wu. Towards More Effective Responsible Disclosure for Vulnerability Research. In *Workshop on Ethics in Computer Security (EthiCS)*, 2023.

[9] Belkin. Belkin Coordinated Vulnerability Disclosure Program. https://www.belkin.com/security-page.html. (Accessed on 2025-10-23).

[10] Rainer Böhme, Lisa Eckey, Tyler Moore, Neha Narula, Tim Ruffing, and Aviv Zohar. Responsible Vulnerability Disclosure in Cryptocurrencies. *Communications of the ACM*, 63(10):62–71, 2020.

[11] Conner Bradley and David Barrera. Escaping Vendor Mortality: A New Paradigm for Extending IoT Device Longevity. In *New Security Paradigms Workshop*. Association for Computing Machinery, 2023.

[12] Bugcrowd. Bug Reporting Guide. https://docs.bugcrowd.com/researchers/reporting-managing-submissions/reporting-a-bug/. (Accessed on 2025-10-29).

[13] Bugcrowd. Bugcrowd Standard Disclosure Terms. https://www.bugcrowd.com/resources/hacker-resources/standard-disclosure-terms/. (Accessed on 2025-10-29).

[14] bugcrowd. Products: Vulnerability Disclosure. https://www.bugcrowd.com/products/vulnerability-disclosure/.

[15] bugcrowd. Reporting a Bug. https://docs.bugcrowd.com/researchers/reporting-managing-submissions/reporting-a-bug/.

[16] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. Emerging Issues in Responsible Vulnerability Disclosure. In *Workshop on the Economics of Information Security (WEIS)*, 2005.

[17] Hasan Cavusoglu, Huseyin Cavusoglu, and Srinivasan Raghunathan. Efficiency of Vulnerability Disclosure Mechanisms to Disseminate Vulnerability Knowledge. *IEEE Transactions on Software Engineering*, 33(3):171–185, 2007.

[18] Daming D Chen, Maverick Woo, David Brumley, and Manuel Egele. Towards Automated Dynamic Analysis for Linux-based Embedded Firmware. In *Symposium on Network and Distributed System Security (NDSS)*, 2016.

[19] Abraham A Clements, Eric Gustafson, Tobias Scharnowski, Paul Grosen, David Fritz, Christopher Kruegel, Giovanni Vigna, Saurabh Bagchi, and Mathias Payer. HALucinator: Firmware Re-hosting Through Abstraction Layer Emulation. In *USENIX Security Symposium*, 2020.

[20] D-Link. DCS-930L/DCS-931L/DCS-932L/DCS-933L/DCS-934L/DCS-5009L/DCS-5010L/DCS-5020L/DCS-5025L/DCS-5030L :: CVE-2019-10999 :: Stack Buffer Overflow. https://supportannouncement.us.dlink.com/announcement/publication.aspx?name=sap10131. (Accessed on 2025-10-29).

[21] D-Link. D-Link End-of-Life Policy. https://www.dlink.com/en/end-of-life-policy. (Accessed on 2025-10-23).

[22] D-Link. D-Link Report Vulnerability. https://support.dlink.com/ReportVulnerabilities.aspx. (Accessed on 2025-11-13).

[23] Aaron Yi Ding, Gianluca Limon De Jesus, and Marijn Janssen. Ethical Hacking for Boosting IoT Vulnerability Management: A First Look into Bug Bounty Programs and Responsible Disclosure. In *International Conference on Telecommunications and Remote Sensing*, 2019.

[24] eBay. https://www.ebay.com/sch/i.html?_nkw=router. (Accessed on 2025-10-30).

[25] Andrew Fasano, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele, Aurélien Francillon, Long Lu, Nick Gregory, Davide Balzarotti, and William Robertson. SoK: Enabling Security Analyses of Embedded Systems via Rehosting. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2021.

[26] FCC Report Database Report/Search Tool for FCC Information. https://fcc.report/. (Accessed on 2025-11-4).

[27] Wil Gibbs, Arvind S Raj, Jayakrishna Menon Vadayath, Hui Jun Tay, Justin Miller, Akshay Ajayan, Zion Leonahenahe Basque, Audrey Dutcher, Fangzhou Dong, Xavier Maso, Giovanni Vigna, Christopher Kruegel, Adam Doupé, Yan Shoshitaishvili, and Ruoyu Wang. Operation Mango: Scalable Discovery of Taint-Style Vulnerabilities in Binary Firmware Services. In *USENIX Security Symposium*, 2024.

[28] GitHub. https://github.com/. (Accessed on 2025-10-30).

[29] Tiina Havana. Communication in the Software Vulnerability Reporting Process. Master's thesis, University of Jyväskylä, 2003.

[30] Alfred Charles Hobbs. *Rudimentary Treatise on the Construction of Locks*, volume 83 of *Weale's Rudimentary Scientific & Technical Series*. J. Weale, London, 1853.

[31] Allen D Householder, Garret Wassermann, Art Manion, and Chris King. The CERT Guide to Coordinated Vulnerability Disclosure. Technical report, Carnegie Mellon University, 2017.

[32] IEEE S&P. Call for Papers. https://sp2026.ieee-security.org/cfpapers.html, 2026.

[33] Mingeun Kim, Dongkwan Kim, Eunsoo Kim, Suryeon Kim, Yeongjin Jang, and Yongdae Kim. FirmAE: Towards Large-Scale Emulation of IoT Firmware for Dynamic Analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2020.

[34] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *USENIX Security Symposium*, 2023.

[35] Linksys End-of-Life and End-of-support Products. https://www.linksys.com/pages/linksys-product-end-of-life. (Accessed on 2025-11-5).

[36] Linksys. Linksys Product Security. https://www.linksys.com/pages/security. (Accessed on 2025-11-13).

[37] Shuhan Liu, Jiayuan Zhou, Xing Hu, Filipe Roseiro Cogo, Xin Xia, and Xiaohu Yang. An Empirical Study on Vulnerability Disclosure Management of Open Source Software Systems. *ACM Transactions on Software Engineering and Methodology*, 34(7), 2025.

[38] Giovane C. M. Moura and John Heidemann. Vulnerability Disclosure Considered Stressful. *ACM SIGCOMM Computer Communication Review*, 53(2):2–10, 2023.

[39] Asuka Nakajima, Takuya Watanabe, Eitaro Shioji, Mitsuaki Akiyama, and Maverick Woo. A Pilot Study on Consumer IoT Device Vulnerability Disclosure and Patch Release in Japan and the United States. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2019.

[40] NDSS. Call for Papers. https://www.ndss-symposium.org/ndss2026/submissions/call-for-papers/, 2026.

[41] Netgear. End of Service - Consumer Products. https://www.netgear.com/about/eos/. (Accessed on 2025-10-29).

[42] Netgear. Netgear Product Security. https://www.bugcrowd.com/resources/hacker-resources/standard-disclosure-terms/. (Accessed on 2025-10-29).

[43] NIST National Vulnerability Database. CVE-2022-3561 Detail. https://nvd.nist.gov/vuln/detail/cve-2022-3561. (Accessed on 2025-10-29).

[44] NIST National Vulnerability Database. CVE-2024-22651 Detail . https://nvd.nist.gov/vuln/detail/cve-2024-22651. (Accessed on 2025-10-29).

[45] NIST National Vulnerability Database. CVE-2024-26342 Detail. https://nvd.nist.gov/vuln/detail/CVE-2024-26342. (Accessed on 2025-10-29).

[46] OffSec. The Exploit Database. https://www.exploit-db.com/. (Accessed on 2025-10-30).

[47] Sandra Rivera Perez, Michel Van Eeten, and Carlos H. Ganan. Patchy Performance? Uncovering the Vulnerability Management Practices of IoT-Centric Vendors . In *IEEE Symposium on Security and Privacy (S&P)*, 2024.

[48] Nilo Redini, Aravind Machiry, Ruoyu Wang, Chad Spensky, Andrea Continella, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Karonte: Detecting Insecure Multi-binary Interactions in Embedded Firmware. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[49] Tobias Scharnowski, Nils Bars, Moritz Schloegel, Eric Gustafson, Marius Muench, Giovanni Vigna, Christopher Kruegel, Thorsten Holz, and Ali Abbasi. Fuzzware: Using Precise MMIO Modeling for Effective Firmware Fuzzing. In *USENIX Security Symposium*, 2022.

[50] Jayashree Srinivasan, Sai Ritvik Tanksalkar, Paschal C Amusuo, James C Davis, and Aravind Machiry. Towards Rehosting Embedded Applications as Linux Applications. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2023.

[51] Prashast Srivastava, Hui Peng, Jiahao Li, Hamed Okhravi, Howard Shrobe, and Mathias Payer. FirmFuzz: Automated IoT Firmware Introspection and Analysis. In *International ACM Workshop on Security and Privacy for the Internet-of-Things*, 2019.

[52] Tay, Zeng, Vadayath, Raj, Dutcher, Reddy, Gibbs, Basque, Dong, Smith, Doupé, Bao, Shoshitaishvili, and Wang. Greenhouse: Single-Service Rehosting of Linux-Based Firmware Binaries in User-Space Emulation. In *USENIX Security Symposium*, 2023.

[53] TechInfoDepot Wiki. https://techinfodepot.shoutwiki.com. (Accessed on 2025-11-4).

[54] threat9. Metasploit: The World's Most Used Penetration Testing Framework. https://www.metasploit.com/. (Accessed on 2025-10-27).

[55] threat9. Routersploit: Exploitation Framework for Embedded Devices. https://github.com/threat9/routersploit. (Accessed on 2025-10-27).

[56] TP-Link. Download for TD-8811 V1. https://www.tp-link.com/us/support/download/td-8811/v1/. (Accessed on 2025-10-01).

[57] TP-Link. TP-Link Product Security Advisory. https://www.tp-link.com/us/press/security-advisory/. (Accessed on 2025-10-23).

[58] USENIX Security. Call for Papers. https://www.usenix.org/conference/usenixsecurity26/call-for-papers#ethics, 2026.

[59] Dingding Wang, Muhui Jiang, Rui Chang, Yajin Zhou, Hexiang Wang, Baolei Hou, Lei Wu, and Xiapu Luo. An Empirical Study on the Insecurity of End-of-Life (EoL) IoT Devices. *IEEE Transactions on Dependable and Secure Computing*, 21(4):3501–3514, 2024.

[60] Yaowen Zheng, Yuekang Li, Cen Zhang, Hongsong Zhu, Yang Liu, and Limin Sun. Efficient Greybox Fuzzing of Applications in Linux-Based IoT Devices via Enhanced User-Mode Emulation. In *ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2022.

[61] Wei Zhou, Le Guan, Peng Liu, and Yuqing Zhang. Automatic Firmware Emulation through Invalidity-guided Knowledge Inference. In *USENIX Security Symposium*, 2021.

[62] ZoomEye Search Engine for Internet Asset Discovery. https://www.zoomeye.ai/. (Accessed on 2025-10-30).

[63] ZyXel. ZyXel Reporting a Security Vulnerability. https://www.zyxel.com/global/en/support/security-vulnerability. (Accessed on 2025-11-13).

Appendix A. Criteria for Creating Additional Exploits

Our additional exploit scripts for empirical testing were selected based on the following criteria:

1) The exploit is publicly available and linked to an existing CVE report.
2) The exploit was released between 2019-2025.
3) The exploit can be recreated by an average programmer in less than a day.
4) The exploit was verified to work on at least one rehosted device image in our dataset.

To fulfill Criteria 1) and 2), our exploits were curated from the National Vulnerability Database, a reputable and easy to use third-party source for CVE reports. For criterion 3) and 4), we intentionally chose exploits that were recent and easy to export as part of our threat model. Our intent is to reflect how public disclosure of a vulnerability can be abused, making it easy for a reasonably-skilled malicious actor to find, create and redeploy known exploits against their choice of known (or unknown) targets.

Appendix B. Crash Detection for Exploit Replay against Rehosted Targets

Each rehosted target was run inside an isolated docker container, while a separate script ran routersploit against a list of known IP addresses for that device. Between each exploit script, we made sure to remove the previous container and images and rebuild the docker container afresh, to avoid artifacts from a previous attempt affecting the current one.

Determining whether a crash occurred followed three checks. If an image-exploit pair fulfilled any of these checks, it was added to a list of potential true positives. We describe how these true positives are filtered in more detail in Section 5.2.

1) RCE command injection.
2) Segmentation Fault.
3) Denial-of-Service.

Check 1) looks at RCE scripts that performed an actual command injection as part of their attack. To make it easier for an automated script check an injection occurred, we had the RCE inject a command to generate a unique file inside the emulated filesystem. Our instrumentation script could then check the root filesystem for the presence of this file to verify that the attack worked.

Check 2) used segmentation faults as the primary indicator that a memory corruption vulnerability occurred. Using the same instrumentation script as Check 1), we check if a *.core* dump file was created. In addition, we modified the qemu binary used to emulate our targets architecture to create a SIGSEGV_HAPPENED file in the root directory if the qemu emulation itself detected a segmentation fault, which we also check for.

Check 3), denial-of-service, is simply the case where the server stops responding entirely. We run two sets of curl commands - one before the executing the exploit, and one after. If both commands return a non-empty webpage within a timeout of five seconds, we assume that the server is still running after the exploit. If the first command returned a non-empty webpage, but the second times out, we treat it

as an indicator that a denial-of-service attack has occurred. This accounts for the case where the exploit does cause a crash, but is undetectable by our first two checks due to the firmware itself blocking access to certain file descriptors.

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper investigates the effectiveness of the responsible disclosure process in the Internet-of-Things (IoT) ecosystem. By developing an automated pipeline called BucketLeak to replay exploits against rehosted firmware, the authors quantify the prevalence of "undisclosed N-days"—vulnerabilities that are publicly known but remain unpatched in many affected device models. The study analyzes over 3,500 firmware images and identifies over one million active real-world devices that remain vulnerable due to incomplete vendor disclosures.

C.2. Scientific Contributions

- Independent Confirmation of Important Results with Limited Prior Research.
- Provides a New Data Set For Public Use.
- Creates a New Tool to Enable Future Science.
- Identifies an Impactful Vulnerability.
- Provides a Valuable Step Forward in an Established Field.
- Establishes a New Research Direction.

C.3. Reasons for Acceptance

1) The paper identifies an impactful vulnerability. By analyzing the gap between vulnerability disclosure and comprehensive patching, the authors highlight a critical systemic failure where malicious actors are effectively provided "keys" to overlooked devices.
2) The paper provides a valuable step forward in an established field. While security problems in IoT are well-documented, this work is the first to rigorously quantify how the disclosure process itself contributes to a "vulnerability amplification" effect across similar device models.
3) The paper creates a new tool to enable future science. The BucketLeak system demonstrates an efficient methodology for scaling firmware analysis through rehosting, which reviewers noted could be applied to other classes of devices.
4) The authors addressed ethical and methodological concerns. In response to the Research Ethics Committee (REC), the authors provided a clear status update on their ongoing responsible disclosure efforts with seven different vendors. They also committed to providing more conservative estimates and confidence intervals for their findings in the final version.