FTP: The Forgotten Cloud

Drew Springall Zakir Durumeric J. Alex Halderman
University of Michigan
{aaspring, zakir, jhalderm}@umich.edu

***Abstract*— Once pervasive, the File Transfer Protocol (FTP) has been largely supplanted by HTTP, SCP, and BitTorrent for transferring data between hosts. Yet, in a comprehensive analysis of the FTP ecosystem as of 2015, we find that there are still more than 13 million FTP servers in the IPv4 address space, 1.1 million of which allow "anonymous" (public) access. These anonymous FTP servers leak sensitive information, such as tax documents and cryptographic secrets. More than 20,000 FTP servers allow public write access, which has facilitated malicious actors' use of free storage as well as malware deployment and click-fraud attacks. We further investigate real-world attacks by deploying eight FTP honeypots, shedding light on how attackers are abusing and exploiting vulnerable servers. We conclude with lessons and recommendations for securing FTP.**

I. Introduction

The File Transfer Protocol (FTP), first introduced nearly 45 years ago [7], was for decades the protocol of choice for moving files between hosts and for distributing them to the world [17]. In recent years, FTP has been largely eclipsed by newer protocols such as HTTP, SCP, and BitTorrent — all of which have received vastly more attention from security researchers. Nevertheless, as of 2015, FTP remains in use by millions of servers that offer more than half a billion files to the public: it is largely forgotten but far from gone.

We present the first comprehensive security analysis on how FTP is used and abused in modern practice. We begin by using Internet-wide scanning to characterize the contemporary FTP server ecosystem, which we find consists of 13.8 million servers in the public IPv4 address space[1]. Of these servers, 1.1 million (8%) permit anonymous logins, making their contents accessible to the public at large. Although many of these publicly accessible servers are operated by large hosting providers, a substantial fraction are consumer devices that provide remote access to data and appear to be mistakenly configured to allow public access.

Publicly accessible FTP sites host an alarming number and variety of sensitive files, which suggests that misconfiguration is widespread. To measure the scope of this phenomenon, we construct a robust FTP enumerator and use it to collect the directory listings of over a million anonymous FTP sites. Since the protocol has accumulated layers of ad hoc extensions over the years, this is challenging to automate at large scale. Nevertheless, our toolchain is able to collect listings of over 600 million files and directories, which range from financial information to email archives, password databases, private keys, and personal photographs. Startlingly, we find that nearly 5% of anonymous FTP servers appear to expose at least one such sensitive file.

[1]In comparison, about 30M hosts complete a TLS handshake on TCP/443, but these have been studied far more extensively (e.g., [21], [30], [32]).

Beyond information exposure, FTP is prone to abuse by malicious parties who seek to attack the server or use it to exploit other systems. Drawing on data from our FTP enumerator and from a series of honeypots, we uncover evidence of several malicious campaigns that leverage anonymous FTP servers to distribute malware, launch DDoS attacks, and carry out SEO campaigns. Among our findings is that more than 20K FTP servers allow anonymous users to write data — which malicious actors are using to deploy malware and trade illicit files — and more than 140K fail to properly validate `PORT` commands — which can be used to probe remote, third-party servers. We also find that nearly 10% of FTP servers listening on public IP addresses report software versions that are susceptible to one or more publicly disclosed vulnerabilities.

With regard to the security provided by FTPS (a protocol extension which allows FTP connections to communicate over TLS), we analyzed the adoption rate as well as how it is being implemented. We find only 793K certificates are in use across 3.4 million servers who support FTPS. While a large number of these are shared-hosting providers, we also find evidence that embedded device manufacturers are shipping identical FTPS certificates and private keys built-in to their devices.

We conclude by attempting to distill the root causes of FTP's persistent vulnerability, by offering potential solutions to improve the FTP ecosystem, and by drawing lessons about user-centered security issues that apply even beyond FTP. Along with these solutions, we analyze possible methods of encouraging their deployment.

FTP is a product of a time when security was much less of a focus than it is on today's Internet. Although the protocol continues to be implemented and deployed, the FTP ecosystem as a whole has only marginally advanced in terms of security. While the existence of FTP-related vulnerabilities may not come as a surprise, the vast number of vulnerable systems and sensitive files — and their persistence up to the present — is shocking. Our study presents a dismal portrait of how FTP is deployed in 2015, but we hope that by shedding light on these ongoing vulnerabilities, the network security community can begin to address them.

II. Background

FTP was introduced in 1971 to allow users to transfer files between network hosts [7]. Clients send text requests in the form of "`<Command> [arguments]\r\n`" to the server and extract a three-digit return code and other request-dependent information from the server's response to determine whether the request was successful. In a typical scenario, a client initiates a connection on TCP/21, and, after receiving a "banner" containing arbitrary text from the server, logs in with the `USER` and `PASS` commands. Once authenticated, the client can list and traverse the accessible

directory structure and upload and download files (depending on the permissions set by the administrator).

In a peculiarity of the protocol, FTP requires two connections: one for control messages and one for transferring the requested data. In traditional *active* FTP, the client sends the `PORT` command whose arguments indicate the client's IP address and an open, ephemeral port that the server should connect back to using a second connection. Unfortunately, this is incompatible with many firewalls and NATs, which are unable to detect that the inbound connection is associated with the original outgoing FTP connection. To address this, *passive* FTP was introduced, in which the client sends the `PASV` command and the server responds with and listens on an ephemeral port that the client opens a second connection to [6]. Regardless of whether the connection is active or passive, the client can then send requests via the control connection to list directory contents and retrieve or store files, which are transmitted via the secondary (`PORT`/`PASV` negotiated) connection.

Although FTP provided an easy and efficient way to transfer files, the mandatory authentication hindered publicly posting data. To address this, the protocol was extended to support *anonymous* FTP, which allows administrators to explicitly allow public access. To use anonymous FTP, the client authenticates with the username "`anonymous`" and their contact e-mail address as the password (if the server requires one). A server configured to allow anonymous FTP will accept any password for the anonymous user [17].

As with many early protocols, FTP was designed with only minimal consideration for security, such that both commands and data are sent in unauthenticated, unencrypted form. To address this, FTPS was introduced [3], [26]. FTPS allows the endpoints to upgrade the connection to TLS, akin to STARTTLS for SMTP. The client sends the `AUTH SSL` or `AUTH TLS` request to the server and reads the response to determine if TLS is supported. If so, the client and servers complete a standard TLS handshake and then continue with the FTP protocol over the secure connection.

This patchwork of extensions — some described in RFCs and some not — has resulted in diverse behavior by different FTP implementations. Server responses to the `USER` login request are a prime example. The return code `331` has at least four meanings depending on the implementation- and language-specific text that accompanies it: "User accepted, send password", "User rejected", "Send virtual-site hostname with username", or "FTPS required prior to login".

While other protocols might fragment under such loose standardization, FTP has been surprisingly resilient. Barebones FTP clients are capable of talking to most server implementations. This is largely due to the human-centered nature of the protocol. FTP clients perform very little of the "heavy lifting" for the user and mainly serve to make FTP communication less tedious. For many operations, replacing a console-based FTP client with a bare TCP connection would result in little additional work for the user.

Unfortunately, this level of user control is an obstacle to large-scale automation and to our goal of analyzing FTP behavior on an Internet-wide basis. In order to study the FTP ecosystem as a whole, we needed to build tools that could carry out all actions autonomously while being robust enough to correctly communicate with diverse real-world implementations.

III. Methodology

To survey FTP at Internet scale, we needed to address three main challenges. The first was how to automate handling of the FTP protocol, with its quirks and myriad implementations. We adopted a reverse-engineering perspective: starting from a simple enumerator that implemented a minimal subset of the FTP protocol, we began testing on a local testbed that consisted of a diverse collection of server implementations. After ensuring correct behavior on our testbed, we tested against gradually larger random samples of live servers. By iteratively expanding the capabilities of the enumerator and reactively adjusting its behavior to oddities found in the wild, we attained a good balance of RFC correctness and compatibility with real implementations.

The second challenge was how to efficiently collect data from FTP servers throughout the IPv4 address space. We adopted a methodology based on the ZMap toolchain [23] coupled with our custom FTP enumerator. In the first stage of our data collection, we used ZMap to perform a host discovery scan on TCP port 21. We then used our enumerator to perform a follow-up connection to each responsive host, attempt an anonymous login (per RFC 1635 [17]), parse each host's `robots.txt` file, and traverse the host's directory structure in a breadth-first manner. Once we finished traversing any publicly accessible directories, we collected the data returned by the `HELP`, `FEAT`, and `SITE` commands. Regardless of whether the server allowed anonymous access, we attempted to initiate a TLS session prior to disconnecting to collect the server's SSL certificate. Our enumerator is written in C using the `libevent` framework [36] and is publicly available at https://github.com/aaspring/ftp-enumerator.

The last main challenge was how to process and analyze the resulting data, which is largely unstructured. Server banners contain arbitrary text, users name files in varying manners and in different languages, and, in some cases, filenames may not describe file content. In order to sift through all this data and establish lower bounds on vulnerability and data exposure, we iteratively processed the dataset, manually selecting and investigating specific evidence of abuse or accidentally exposed data. After each iteration, we measured the number of servers displaying the same or similar behavior. Although this provides an estimate of the range and scope of vulnerability, it may result in the statistics we report underestimating the true scale of the problems.

Since many files appeared to contain sensitive data that was inadvertently made public, we chose not to download files in bulk using our enumerator. For purposes of high-level statistics, we attempted to infer file contents based on the filename and extension. Without attempting downloads, we cannot determine with certainty whether the anonymous FTP user has permission to read the files. To address this, we examined the all-users permission in directory listings to determine whether an anonymous user could likely retrieve each specific file. In cases where the server did not display permissions (as with most Windows-based servers), we labeled the files as "unk-readability".

A. Ethical Considerations

As with any research conducted through Internet-wide scanning, our work raises important ethical considerations. We carefully considered the impact of our experimental measurements on parties ranging from our local institutional network to the owners of remote systems, and we took numerous steps to prevent or mitigate potential harms.

When scanning for FTP sites, we followed the recommendations set forth by Durumeric et al. [23]. We coordinated with our local network administrators and upstream ISP to ensure that our scans did not adversely impact network operations. We signaled the benign intent of our scanning hosts by setting descriptive WHOIS records and reverse DNS entries for them and posting a simple website on port 80 that described the goals of the research, including what data we collected and how to contact us. We invited user exclusion requests and responded to requests within 24 hours, and we preemptively excluded any hosts that our institution had previously been asked to exclude from scanning research as part of other studies.

When logging into FTP servers, we never attempted to guess login credentials or to exploit vulnerabilities to access non-public data. We also made a concerted effort to parse FTP banners for messages stating that the server did not permit anonymous access and discontinued the login attempt in that case. We strictly followed RFC 1635 ("How to Use Anonymous FTP") [17]. If the server required a password for the anonymous login, we sent our team's abuse contact email address.

When traversing sites, we followed the community's Robots Exclusion Standard, fetching each host's `robots.txt` file, if present, and following it per Google's specification [29]. To ensure that we did not inundate a server with requests, we spread concurrent connections across a large number of widely dispersed hosts, and we limited the speed of interactions with each host to two requests per second. We also imposed a maximum of 500 requests per connection. If the server terminated the connection at any point during directory traversal, we interpreted this as an explicit refusal of service and ceased interaction with that server.

As we will discuss in the remainder of the paper, we were surprised to find that a significant fraction of the data available via anonymous FTP appears to have been inadvertently published. For this reason, we stopped short of downloading files except in a few particular instances as necessary for verification and even then only after careful deliberation and consultation with colleagues. Despite the fact that these files and directory listings are publicly accessible, there would be significant risk in publishing an exhaustive list of files that could then be then be trivially retrieved and potentially abused. As such, we do not intend to publish our enumeration dataset. We are working to notify responsible entities in likely instances of sensitive information disclosure.

IV. FTP Landscape

Between June 18 and 21, 2015, we performed a scan of the IPv4 address space and enumerated publicly accessible FTP servers. In this scan, 21.8M hosts responded on port 21 and 13.8M sent an FTP-compliant banner. Of these, 1.1M (8%) allowed anonymous access (see Table I). Of the servers that allowed anonymous access, 268K (24%) exposed some form of data; the remainder contained empty or inaccessible directories. We detected `robots.txt` files on 11.3K servers. Of these, 5.9K instructed our crawler to exclude the entire filesystem, which we adhered to. On 26.7K servers, the file structure required more than 500 requests to fully traverse, and so our enumeration did not reach the entire accessible filesystem.

TABLE I. General metrics from FTP enumeration

IPs scanned	3,684,755,175	(85.79% of IPv4 address space)
Open port 21	21,832,903	(0.59% of scanned IPs)
FTP servers	13,789,641	(63.16% of IPs with port open)
Anonymous FTP servers	1,123,326	(8.15% of responsive FTP servers)

TABLE II. Breakout of servers in each category

Server Classification	All FTP Servers	Anonymous FTP Servers
Generic Server	5,957,969 (43.21%)	704,276 (62.66%)
Hosted Server	1,795,596 (13.02%)	174,198 (15.50%)
Embedded Server	1,786,656 (12.95%)	93,484 (8.32%)
Unknown	4,249,417 (30.82%)	151,927 (13.52%)

TABLE III. ASes accounting for 50% of all FTP types

AS Type	All FTP (78)	Anonymous FTP (42)
Hosting	50	29
ISP	25	11
Academic	3	2

TABLE IV. Classes of embedded devices

Device Type	All FTP	Anonymous FTP
NAS	198,381	18,116
Home Router (user-deployed)	59,944	6,788
Printers	62,567	60,771

A. Who is Deploying FTP?

We find that FTP servers and anonymous FTP servers are both widespread, with 34.7K ASes containing FTP servers and 16.4K ASes containing anonymous FTP servers. While FTP servers are present in a large number of networks, a few ASes contain the bulk of these servers: 78 ASes account for 50% of FTP servers, and 42 ASes account for 50% of anonymous FTP servers (see Figure 1).

To better understand why servers allow anonymous FTP access in so many networks, we categorized the networks that contain a large number of anonymous FTP servers and also developed fingerprints to identify specific implementations and devices based on banner, certificate, and implementation-specific responses. With these fingerprints, we were able to classify 69% of all FTP servers and 86% of anonymous FTP servers into one of several broad categories: (1) large shared-hosting providers, (2) provider-deployed embedded devices (e.g., cable modems), (3) consumer-deployed devices (e.g., home NAS devices), and (4) generic servers (i.e., widespread implementations that we were not able to categorize further). We show the breakdown in Table II. We note that our estimates in the "Hosted Server" and "Embedded Server" categories are lower bounds, since some devices and services do not expose identifiers that allow for more specific classification.

However, even the lower bounds demonstrate that the problems are systemic.

Shared Hosting Providers. As shown in Table III, IP addresses within large hosting provider ASes account for a substantial portion of both FTP and anonymous FTP servers. 50 of the largest 78 FTP ASes and 29 of the 42 largest anonymous FTP ASes are owned by large hosting providers that offer shared hosting, virtual private server (VPS), co-location, and/or private cloud services. Table VI shows more details on the most commonly seen ASes.

These providers appear to deploy FTP as part of their default service offering, which, anecdotal evidence suggests, are managed using web hosting automation software such as cPanel or Plesk. In total—combining the hosts from the large providers we identified and our fingerprints—we estimate that at least 4.6M FTP servers (34%) and 470K anonymous FTP servers (42%) belong to hosting providers. In the largest case, home.pl (a large Polish hosting provider), approximately two-thirds of all advertised IPs support FTP, and more than half support anonymous FTP. However, despite the large raw numbers, we find that, in general, there is little sensitive user data exposed on these hosts compared to personal devices.

Provider Deployed Embedded Devices. As can be seen in Table III, all but a handful of the remaining large ASes belong to ISPs. In many cases, service providers have deployed embedded devices, such as DSL modems, all of which have FTP enabled. In the most common case, more than 150K FRITZ!Box DSL modems are found. The modems are primarily located in Germany and have been deployed en masse by Deutsche Telekom. We also frequently saw devices from ZyXEL, AXIS, ZTE, and others (see Table V). A negligible fraction of the devices had anonymous FTP enabled.

Consumer Embedded Devices. Most worryingly, we find that hundreds of thousands of consumer devices have FTP enabled, and over 93K of these devices allow anonymous access. These devices are largely network attached storage (NAS) appliances, home wireless routers, and printers. These home devices are from a variety of manufacturers, including Buffalo, Asus, Xerox, Dell, Ricoh, and Synology. We present the most common devices in Table VII.

For three of the devices we identified, more than 98% of visible hosts have anonymous FTP enabled. While it is plausible that a few home users are intentionally enabling anonymous FTP to publish data to the world, that such a large fraction are doing so seems unlikely. Instead, we suspect that these particular devices enable anonymous FTP by default, or that they are designed in a way that leads users to misunderstand the risks of enabling certain features.

TABLE V. **COMMON PROVIDER DEPLOYED DEVICES** — WE FIND THAT INTERNET SERVICE PROVIDERS FREQUENTLY DEPLOY DEVICES WITH FTP EN MASSE. VERY FEW HAVE ANONYMOUS ACCESS ENABLED.

Device	# Found	# Anonymous
FRITZ!Box DSL modem	152,520	49 (0.03%)
ZyXEL DSL Modem	29,376	1 (0.00%)
AXIS Physical Security Device	20,002	58 (0.29%)
ZTE WiMax Router	14,245	0 (0.00%)
Speedport DSL Modem	13,677	0 (0.00%)
Dreambox Set-top Box	12,298	0 (0.00%)
ZyXEL Unified Security Gateway	11,964	0 (0.00%)
Alcatel Router	10,383	0 (0.00%)
DrayTek Network Devices	4,161	0 (0.00%)

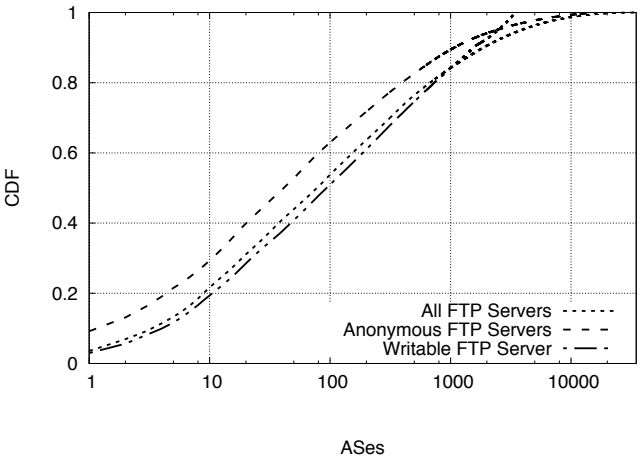

Fig. 1. **Distribution of FTP servers by AS.** CDF of all FTP, anonymous FTP, and writable FTP servers with regard to aggregate number of ASes viewed.

TABLE VII. SAMPLE OF EMBEDDED SERVER DEVICES THAT ARE DEPLOYED AS STANDALONE

Device	# Found	# Anonymous
QNAP Turbo NAS	57,655	1,637 (2.84%)
ASUS wireless routers	52,938	5,891 (11.13%)
Synology NAS devices	43,159	2,942 (6.82%)
Buffalo NAS storage	22,558	8,870 (39.32%)
ZyXEL/MitraStar NAS	9,456	310 (3.28%)
RICOH Printers	8,696	7,606 (87.47%)
LaCie storage	4,558	2,919 (64.04%)
Lexmark Printers	3,908	3,896 (99.69%)
Xerox Printers	3,130	2,906 (92.84%)
Dell Printers	2,555	2,515 (98.43%)
Linksys Wifi Routers	2,174	624 (28.72%)
Lutron HomeWorks Processor	1,006	1,003 (99.70%)
Seagate Storage devices	629	594 (94.44%)

The surprising number of servers that support anonymous FTP raises several important questions: What data has been inadvertently leaked on these devices? In the shared hosting environments, code frequently contains API keys, and other secrets; are these files accessible over anonymous FTP? Are malicious actors taking advantage of these servers? In the next several sections, we present case studies in which we investigate these questions.

V. OVER EXPOSURE

With more than 1.1M FTP servers anonymously accessible, the questions "What data is exposed?" and "Are users explicitly posting this data?" immediately arise. We investigated the directory listings from publicly accessible anonymous FTP servers and found that a large amount of user data is being exposed to the public. In this section, we discuss the files available on anonymous servers and the devices responsible for this data exposure.

A. Data Exposure

Within our dataset, `index.html` is the most common file found with 494K instances on nearly 25K servers. This comports with our previous analysis describing large hosting

TABLE VI. TOP 10 ASES BY NUMBER OF ANONYMOUS FTP SERVERS

AS	IPs advertised	FTP servers	Anonymous FTP servers
AS12824 home.pl S.A.	205,312	136,765 (66.61%)	103,175 (75.44%)
AS46606 Unified Layer	516,864	246,470 (47.69%)	44,273 (17.96%)
AS2914 NTT America, Inc.	7,880,192	298,468 (3.79%)	36,045 (12.08%)
AS20013 CyrusOne LLC	111,360	64,790 (58.18%)	30,772 (47.50%)
AS40676 Psychz Networks	641,024	64,233 (10.02%)	27,507 (42.82%)
AS34011 domainfactory GmbH	93,440	21,153 (22.64%)	19,077 (90.19%)
AS4134 Chinanet	120,757,504	464,384 (0.39%)	18,996 (4.09%)
AS18978 Enzu Inc	727,808	73,541 (10.11%)	17,510 (23.81%)
AS18779 EGIHosting	1,890,304	27,804 (1.43%)	16,329 (58.73%)
AS4766 Korea Telecom	53,733,632	211,479 (0.39%)	16,222 (7.67%)

TABLE VIII. MOST COMMON FILE EXTENSIONS ACROSS KNOWN SOHO DEVICES

Extension	# Files	# Servers
.jpg	15,962,091	10,187
.mp3	2,443,285	4,912
.pdf	1,010,005	9,825
.avi	955,832	4,954
.gif	762,581	5,291
.png	476,530	5,456
.mp4	456,471	5,797
.doc	440,118	3,924
.html	426,646	5,275
.zip	294,649	6,698

providers as a major source of anonymous FTP servers in the IPv4 address space. Table VIII shows the most common file extensions found on devices we can identify as small-office/home-office devices. Although we will later address the use of anonymous FTP for pirated content exchanges, we find that the majority of `.mp3`, `.avi`, and `.mp4` files appear to be users' personal media collections.

Sensitive Documents. We also looked for files from less common yet more sensitive classes of documents. We find large numbers of email and financial documents, including more than 12.6K email archives and 7.7K Quicken data files. We show a breakdown of the most prevalent and likely sensitive files in Table IX. We also found instances of what appear to be company- or office-wide backups on single FTP servers resulting in occurrences of a single server holding 688 `.pst` files (Outlook mailboxes), another with 146 `shadow` files (Unix password databases), and a third with years of backups for financial files. Anecdotally, we observed many other apparent examples of private information while sifting through our dataset, including what appeared to be medical records, companies' bids for work, and human resource records. Unfortunately, these are mostly personalized in terms of filename, location, and (likely) contents, making precise measurement infeasible.

Photo Libraries. A similar pattern emerged for personal photos, where we found 13.7M photos (12.9M with readable permissions) with names consistent with the common default formats for consumer cameras on 17K servers. Based on the directory paths alone, it appears that these photos provide an intimate glimpse into users' personal lives. Weddings, family reunions, vacations, birthday parties, and more are organized and available to any voyeur who wishes to look.

Root File Systems Exposed. Many devices look to expose most of, if not all, of their filesystem over FTP. By looking for known OS-root directories, we can estimate the number. Within the same FTP path, we looked for : `bin`, `var`, `boot`, and `etc` on Linux; `Applications`, `bin`, `var`, `Library` and `Users` on Mac OS; `Program Files`, `Documents and Settings`, and `WINDOWS` on early Windows versions; and `Windows`, `Program Files`, and `Users` on later Windows. With this approach we found 825 Windows, 3,858 Linux, and 15 OS X servers with their OS-root exposed.

Scripting Source Code. In the case of hosting providers, allowing customers to upload and traverse webpages' source files is not necessarily a vulnerability, but if it exposes the source code to server-side scripts, it can drastically reduce the difficulty of finding vulnerabilities to exploit. Additionally, any inline configuration files (such as `.htaccess`) and any secrets (such as API keys) contained within server-side code are also shared. In total, we found 189.4K `.htaccess` files on 4.5K servers along with 10.2M other server-side scripting source files on 32K servers.

B. Responsible Devices

When we look to explain the exposure of this sensitive information, our device fingerprints described in Section IV shed some light on possible sources. As shown in Table X, we are able to determine 12.3% of the devices exposing sensitive user information, pointing to a manufacturer and often a specific device model. We see two major classes of devices: consumer NAS devices and consumer-grade "smart" routers. Using this information, we investigated the user manuals for many of these devices to determine how and why anonymous FTP might be enabled and exposed to the Internet.

Consumer NAS Devices. The first of these major classes is the personal/small-office Network Attached Storage (NAS) devices previously referenced in Section IV-A and includes 198K devices (18K of which enable anonymous access). Similar in form to USB external disks enclosures, these devices contain one or more hard drives and connect to the network via ethernet or WiFi.

The user manuals for many of these devices provide great insight as to why they are accessible from the Internet. While we, as security researchers, recommend that firewalls or NAT be used to restrict access to user's internal/trusted home networks, it is apparently not the recommendation of device manufacturers. Many of the manuals contain dedicated sections on *Accessing your device from anywhere* where they describe how to set up port forwarding or UPnP sharing on the user's home router, how to find the router's public IP address, and information about Dynamic DNS providers [12], [33], [40]. Many of

TABLE IX. EXAMPLES OF SENSITIVE EXPOSURE VIA ANONYMOUS FTP INCLUDING FILE PERMISSIONS

Type	File	# Servers	# Files	# Readable	# Non-readable	# Unk-readable
Financial Information	TurboTax Export	464	8,190	8,139	6	45
	Quicken Data	440	7,702	7,652	6	241
Password Databases	KeePass/KeePassX	210	1,812	1,762	6	44
	1Password	11	24	23	0	1
Key Material	SSH host private keys	819	1,597	139	1,427	31
	Putty SSH client keys	82	128	98	0	30
	"priv" `.pem` files	701	1,397	1,335	2	60
Other	`shadow` files	590	718	238	473	7
	`.pst` files	2,419	12,636	10,918	103	1,615

TABLE X. BREAKOUT OF DEVICES EXPOSING USER INFORMATION

Type of Exposure	Generic	Embedded			Hosting	Unk
		NAS	Router	Other		
Sensitive Documents	26.29%	7.08%	20.16%	0.18%	0.12%	45.54%
Photo Libraries	39.98%	12.35%	11.52%	0.01%	3.12%	33.00%
Root File Systems	10.54%	0.68%	1.30%	0.00%	0.00%	87.34%
Scripting Source	72.51%	1.74%	3.26%	2.36%	3.48%	16.56%
All	56.05%	4.54%	6.31%	1.45%	3.00%	28.67%

devices display their IP address in their FTP banners, and we can see from our dataset that many are using private addresses (192.168.0.0/16, 10.0.0.0/8, etc.), indicating that they are configured in this way. With few exceptions (e.g., [11]), these manual sections are devoid of warnings on the hazards of making a device accessible from the entire Internet. As we will discuss in Section VIII, even devices without anonymous access enabled are vulnerable to attackers guessing weak or default passwords.

With regard to anonymous access, the user manuals are often ambiguous with regard to permissions and how they apply in different configurations. The LaCie CloudBox's description of the built-in "`Family`" directory is an example of this. While one part of the manual explicitly states,"This type of unrestricted use of a shared folder is called *Public Access*", other portions discuss it in terms of "everyone in the home" or "all the computers connected to the same router", giving the impression that the public access is restricted to the local network [33]. While misleading, this description of "in the home" is not incorrect. The manual typically describes accessing the device via SMB, which is *not* a routable protocol due to it being built on top of NetBIOS. If accessed via FTP, this directory allows anonymous access.

Smart Routers. The second class of devices is "smart" home routers, which, unlike traditional home routers, do much more than provide basic NAT and firewall features. Such devices account for 59.9K devices (6.7K with anonymous access). Among other features, many smart routers include NAS-like functionality and can expose external USB drives over the network [35] [5], and others ship with pre-installed internal hard drives [4].

Unlike NAS devices, home routers are typically on the edge of the user's network and do not require port forwarding to be accessible. Unfortunately, this further lowers the bar for exposing user information. For a time, ASUS routers automatically enabled anonymous access for any USB drive attached to the router [38]. But even without such dangerous default configurations, routers as a whole are no better than NAS devices with regard to their documentation. In addition to the lack of clarity with regard to public access to shares described above, some routers also suffer from a lack of proper naming conventions. For example, certain routers include a built-in firewall rule named, "Filter Anonymous Internet Requests", yet this rule actually only blocks anonymous ICMP echo requests and not anonymous FTP access [34].

VI. MALICIOUS USE

We find several malicious campaigns that leverage world-writable anonymous FTP to distribute malware, launch DDoS attacks, and carry out SEO campaigns. During this investigation, we downloaded a small number of files associated with malware for analysis. In all instances, the files were hand selected based on their frequent occurrence across diverse servers or their frequent co-location with known malicious files.

A. World-writable Determination

We first investigated whether anonymous servers support file upload through FTP or if our read access simply allows us to see infections that originated from an unrelated vulnerability. The most precise way to determine whether a server is world writable would be to attempt to upload a file to every anonymous FTP server. However, to reduce the impact of our scans, we took a more conservative approach and instead looked at the attributes and behavior of writable servers to create a reference set of files that indicate that a server allows anonymous write access.

This set mainly consists of two types of files. The first type is those indicated by the server as having been uploaded anonymously. Some servers are configured to allow anonymous users to upload files, but those files may not be downloaded until the administrator changes the file's ownership. Pure-FTPd explicitly confirms that certain files were uploaded by responding to relevant `RETR` commands with "`This file has been uploaded by an anonymous user. It has not yet been approved for downloading by the site administrators.`" The second type is files that are parts of

write-probing campaigns. The three most pervasive probing campaigns upload files named `w0000000t.[txt/php]`, `sjutd.txt`, and `hello.world.txt`. Each file contains a simple string such as "`Anonymous`", "`test`", random characters, or a small amount of base64-encoded data. It appears that the attackers attempt to upload these probe files prior to uploading their attack payloads.

In addition to these two types, there are also smaller cases where a file was added to the reference set due to the behavior of certain server implementations. Specifically, some configurations allow anonymous users to upload files, but not delete or overwrite existing files. When a second file with the same name is uploaded, an incrementing number is appended to the end of the filename resulting in the set of files "`name`", "`name.1`", "`name.2`", etc. being seen on a server. Although this behavior is heuristically testable, many other files use this format and are not related to FTP or anonymous upload. Therefore, we do not add all files which match this format to our reference set.

As seen in Figure 1, we find 19.4K servers in 3.4K ASes that appear to be world-writable. We note that this is a lower bound given that not all servers will have been found by a write probing campaign or previously had files uploaded anonymously. An attacker's software could fail to handle certain quirks, the attackers may not have targeted the entire IPv4 address space, the flux of IPv4 may have resulted in the attackers (or ourselves) not finding certain servers, or, as we will show in Section VIII, attackers might remove the evidence of their probing.

B. Leveraging Server-side Scripting

As discussed in Section V, many anonymous FTP servers provide anonymous access to the web root. To estimate the number of hosts that serve both FTP and a web server, we used the results of an HTTP scan supplied by Censys [19] from the same time period as our enumeration. In total, 9.0M IP addresses (65.27% of all FTP servers) contain both a web-server and an FTP server. Of these, 2.1M IPs (15.01% of FTP servers) indicate that they support server-side scripting by returning either the "`X-Powered-By: PHP`" or "`X-Powered-By: ASP.NET`" HTTP header.

We find that attackers are aware of this and have been using this overlap for quite some time. The most direct usage we find is that attackers leverage server-side scripting for Remote Access Tools (RATs). These RATs vary from the extremely complicated and fully featured to a single line of "`<?php eval($_POST[5]);?>`". It appears common practice to upload these files across the filesystem to improve the chances of landing in the web root. While we see many different variations, only a few can be sourced to FTP with our world-writable methodology. Restricted to only the RATs in our reference set, we find 6K RAT related files on 724 servers.

We also found evidence of attackers using anonymous FTP to set up UDP DDoS infrastructure. Two campaigns, `history.php` and `phzLtoxn.php`, appear to be the most widespread. Both are simple PHP scripts which receive a target host/port and time length from the GET parameters and send 65kB UDP packets as fast as possible for the specified length of time. We found 1,792 servers infected with these campaigns.

It appears that some campaigns involve multiple stages that upload incrementally more complicated server-side scripts as they gain more information about their new victim. The `ftpchk3` campaign is likely a four-stage campaign that operates in this manner. The first stage is a small `ftpchk3.txt` file which is likely a write probe. The second stage consists of a `ftpchk3.php` script which simply echos the text "OK"—likely to determine whether the attacker can access a directory served by a server-side scripting engine. The third stage is a second `ftpchk3.php` script that gathers information such as PHP version, loaded extensions, and what, if any, Content Management System is used. We believe that there is a fourth stage which uses the information gained to act maliciously, but our dataset does not allow us to conclusively assert what that is. In all, we found 1,264 servers in some state of infection by the `ftpchk3` campaign.

While not within our reference set, one especially interesting find is the Holy Bible SEO campaign. This campaign uses the PHP scripting engine to search the victim's filesystem for web files (e.g., HTML, PHP, and ASP files), in which it injects `href` tags. Additionally, it spreads itself across the filesystem and deletes any file with extensions such as `.bak`, `.zip`, `.apk`, and `.msi`. This campaign is difficult to heuristically search for due to its use of extremely common file names such as `index.php` and its many different versions. Our best heuristic comes from an ancillary file which appears to be the attackers "tag" named `Holy-Bible.html`. This file is not involved in the maliciousness, but appears frequently with the malicious files and shares identifying strings with the malicious files. We find 1,131 servers that contain this file, and 55.35% of these servers also have one or more of our reference set of files that denote that the server is world-writable.

C. Other Campaigns

During our investigation, we also discovered a set of other campaigns. One of the oddest is an apparent advertising campaign for a "really cool software cracking service" [1], in which .pdf and .ps fliers are uploaded to world-writable FTP servers and explain that the proprietors are willing to create keygens and dongle emulators to enable pirating software. Readers are to contact the service via Bitmessage [8] or e-mail, and the service charges either $300 or $500. We found 2,095 servers that contain the campaign's fliers. Ramnit, a botnet, is known to use an anonymous FTP server to give its masters easy access to the victim's filesystem. Although Symantec reports that Ramnit's FTP server uses port 22, we discovered 1,051 FTP servers on TCP/21 that carry the malware's FTP banner of "`220 220 RMNetwork FTP`" [39]. None allowed our enumerator to anonymously authenticate.

We also find attackers sharing "WaReZ" data—pirated software and media. While a large amount of music and movies are available over anonymous FTP, much of it appears to be personal libraries, which are exposed along with other information as described in Section V. But one data transport campaign bears a signature that allows us to identify it. This campaign is identified by the directory names that it uses for storage—2-digit year of upload + 2-digit month + 2-digit day + 6-digit time + "p". Overall, there appear to be 4,868 servers which were at one time used by this campaign, but many of the directories are empty. We do not know if these were abandoned

TABLE XI. NUMBER OF SERVERS VULNERABLE TO CVES

Implementation	Vulnerability	CVSS Score	Number IPs
ProFTPD	CVE-2015-3306	10.0	300,931
	CVE-2013-4359	5.0	24,420
	CVE-2012-6095	1.2	1,098,629
	CVE-2011-4130	9.0	646,072
	CVE-2011-1137	5.0	646,072
Pure-FTPD	CVE-2011-1575	5.8	3,305
	CVE-2011-0418	4.0	3,309
vsFTPD	CVE-2015-1419	5.0	658,767
	CVE-2011-0762	4.0	125,090
Serv-U	CVE-2011-4800	9.0	244,060

upload sites or if the content was deleted after it was shuffled between actors.

VII. CASE STUDY—KNOWN VULNERABILITIES

In this section, we present estimates of the number of servers that are currently subject to exploitation using known vulnerabilities.

A. CVEs

We analyzed the version strings presented in FTP banners and find that more than one million servers are vulnerable to common knownly attacks. We show a breakdown of IPs vulnerable to each CVE in Table XI. For ethical reasons, we did not exploit any vulnerabilities on the hosts.

B. PORT bouncing

As discussed in Section II, the PORT command supplies an IP address and port to the server which then initiates a TCP connection back to that IP (assumed to be the client). If a server does not verify that the requested data channel's client IP address is the same as the control channel's, then the server can be used to connect to a 3rd party on the attacker's behalf. This vulnerability is well known and has been publicly noted by CERT since as early as 1997 [13] and we can easily check whether servers support properly validate the PORT command arguments.

By sending a PORT command specifying a different IP address that we control, we can probe whether each anonymous FTP server is vulnerable. We found 143,073 FTP servers (12.74% of anonymous FTP servers) failed to properly validate PORT parameters and created a TCP connection to an IP address other than the control channel's. The vast majority of these servers (71.5%) are within `AS12824 home.pl S.A.` and likely originate from the home.pl hosting service's default software. In addition, the FileZilla FTP server implementation failed to properly validate PORT commands in all releases from January 1, 2003 until May 6, 2015 [31]. Although not all allowed anonymous access and thereby allowing us to test, we found 409K Filezilla implementations on the IPv4 address space of which the majority are likely exploitable after login.

The most straightforward use of this vulnerability is to perform anonymous port scans. While this is useful to attackers, it can be combined with more subtle techniques to create more powerful attacks. For example, if the FTP server is within an otherwise inaccessible network, an attacker can use the PORT command to port scan the internal network. To measure this, we checked servers' responses to the PASV request and for an IP address different than that which we originally connected to, indicating it was behind a NAT. We found 18,947 servers behind a NAT, of which 846 do not properly validate PORT parameters. Another way to leverage servers that fail to validate PORT arguments is the classic "Bounce Attack" in which the attacker induces the FTP server to conduct application-level interaction with a third-party. For example, the attacker can combine the PORT command with a world writable filesystem to coerce the server to send FTP or SMTP commands to a third party, by uploading a file containing the sequence of commands and then PORT bouncing it to third-party server [3]. We found 1,973 servers which are both world-writable and fail to properly check the parameters of the PORT command.

VIII. CASE STUDY—ONGOING MALICIOUSNESS

To detect other attacks, we ran eight FTP honeypots that expose anonymous, world-writable FTP servers for three months. We worked to be reactionary to attackers' behavior: after observing attackers' attempts to blindly traverse certain file paths, we created those paths and populated them with representative files in order to observer the attackers' behavior the next time they probed our honeypots.

A. Results

In total, we observed 457 unique IP addresses scanning TCP/21. Interestingly, over 30% come from the "China Unicom Henan Province Network" AS. 85 IPs spoke FTP to our honeypots, with most of the remainder attempting to fetch the root webpage via the HTTP GET command. 16 IPs attempted to traverse directories and 21 listed the contents of directories; in both cases, some were blind traversals.

In total, we observed over 1,400 unique username-password combinations used to attempt to authenticate. While most of these were simple attempts at guessing weak passwords, we also saw evidence of attempts with default passwords. We did observe one attempt to exploit CVE-2015-3306 [15], one attempt to exploit a Seagate devices' lack of a root password to upload a RAT [24], and 8 addresses testing whether they could exploit the PORT bounce attack. All eight PORT bounce attempts targeted the same 3rd party IP indicating that they were potentially part of the same campaign.

B. Analysis

Compared to the number of malicious files we found in our crawling, there were surprisingly few attacks on our honey pots. One possible reason for this is the historical nature of FTP. In other words, we may be seeing remnants of past malicious campaigns on servers that remain online today. In one case, we found files consistent with the dsns network scanner [18] from early as 2004. That said, while we do not see many instances of active maliciousness, those that we see are in line with our analysis. We find several campaigns looking for world-writable FTP servers, in which clients attempted to upload and then delete the `hello.world.txt` write probe on our honey-pots. We also observed attackers searching for web-root directories such as `cgi-bin`, `www`, and `public_html`. This further validates that attackers are cognizant of and actively

looking for directories which may be useful for server-site scripting attacks.

We also note 36 IPs issued the `AUTH` command and attempted to complete TLS handshakes, likely to identify devices by certificate. We also see behavior consistent with the WaReZ transporters from Section VI—we observed attempts to create directories with no data uploaded to them.

The questions of whether these attacks are malicious or academic exists. We saw both explicit evidence of both: some IPs were nearly identically configured compared to ours with landing pages describing scanning research while others are testing `PORT` validation from Tor Exit Nodes. Most IPs provide no indication of their intention.

IX. Case Study—FTPS Impact

FTPS was created in an attempt to bolt on security to the FTP protocol. The client signals that it wishes to use FTPS by sending the `AUTH SSL` or `AUTH TLS` request to the server. After the server responds positively, both sides conduct a TLS handshake to secure the connection before continuing with the standard FTP protocol inside of the TLS connection. By conducting this negotiation, users are able to protect their username and password combinations as well as any data being transferred.

During our enumeration, we found that 3.4M FTP servers (25%) support FTPS, but less than 85K require it before authentication. Looking for an explanation for this low rate, we surveyed popular web browsers and built-in command line tools on OS X, Linux, and Windows. To our surprise, we found that not a single one supported FTPS. When a server requires the connection to be secure before accepting the `USER` request, these built-in clients disconnect. Third party FTP clients that support FTPS are available for all three major OSs. Even in the case that both the client and the server support FTPS, the security provided is less than would be expected. 1.7M (50%) of the sites that support FTPS use self-signed certificates, which provide no way for the client to check the identity of the server. Although some third-party clients can pin certificates internally, there still exists a trust-on-first-use authentication vulnerability.

Surprisingly,, there are only 793K unique certificates across all 3.4M servers that support FTPS. This appears to be due to two reasons. First, hosting providers commonly use their browser-trusted wildcard SSL certificate on all shared-hosting servers. Second, device manufacturers are deploying identical certificates on all their devices. We show the most common certificates in Table XII and the most common certificates shipped with devices in Table XIII. Because each of these devices uses the same certificate and key, an adversary could extract the private key from any device to Man-in-the-Middle other connections.

TABLE XII. Top 10 most common FTPS certificates

Certificate CN	# Servers	Browser-trusted?
`*.opentransfer.com`	193,392	Yes
`*.securesites.com`	134,891	Yes
`*.home.pl`	125,197	Yes
`*.bluehost.com`	59,979	Yes
`localhost`	47,887	No – self-signed
`ftp.Serv-U.com`	26,209	No – self-signed
`*.bizmw.com`	26,172	Yes
`*.turnkeywebspace.com`	22,075	Yes
`ispgateway.de`	19,355	No – self-signed
`*.sakura.ne.jp`	17,495	Yes

TABLE XIII. Devices that share FTPS certificates

Device	# Found
QNAP NAS (#1)	11,236
ZyXEL Unk	8,402
Buffalo NAS	7,365
LGE NAS	6,220
Axentra HipServ	2,965
RhinoSoft	1,835
Symon Media Player	606
QNAP NAS (#2)	615
AsusTor NAS	367

X. Discussion

The kinds of vulnerabilities we highlight in this paper are not new. Password-less login vulnerabilities, WaReZ shuffling, and port bouncing have been exploited for practically the life of FTP. Yet we found that a large number of servers are still vulnerable to these problems, and they continue to be actively exploited by malicious actors. Part of the reason for this may be inattention. While security researchers are focused on more modern and widespread protocols, such as HTTPS and Bitcoin, older and more basic protocols such as FTP have not gone away and have not become any more secure.

As we described in Section V, consumer devices with embedded FTP servers appear to be compounding the problem. Users seeking remote access to their data may expose FTP on public IP addresses without understanding the security consequences. Worse, at least some devices have confusingly named options for enabling anonymous FTP, and others appear to come with it enabled by default. We urge device manufacturers to re-examine their defaults, user interfaces, and documentation in order to effectively inform users that these configurations will provide global public access to their data. While usable security is difficult to achieve, research has shown that well designed notifications can improve users' adherence to browser security warnings [25], and consumer server devices may benefit by following a similar strategy. As the Internet of Things continues to expand and the move towards IPv6 continues—increasing the number of world-addressable devices a consumer owns—this will likely become a growing problem.

Some of the most difficult cases to make sense of are devices that come with anonymous FTP enabled by default. It seems that either these manufacturers do not understand the dangers associated with data exposure from anonymous FTP or they are acting with reckless disregard for their customers' security and privacy. Either way, an effective remedy may be to create stronger incentives for manufacturers of consumer devices to take security seriously. This may be an ideal instance for an external organization such as a "CyberUL" to provide a quality certification that a particular device provides at least a minimal amount of security protections to the end-consumer [16]. While creating a comprehensive suite of security tests is still an open research problem, it would be easy to test for well known and often exploited vulnerabilities such as anonymous logins and port bouncing.

XI. Related Work

FTP has been largely ignored by the security measurement community and, to the best of our knowledge, this study is the first that focuses on how anonymous FTP has been deployed in practice. There have, however, been several recent studies that employed Internet-wide scanning, that investigated the security of embedded devices, or that identified how poor cross-protocol interactions can lead to attacks similar to the ones we observed.

Internet-wide Scanning A large number of studies have used Internet-wide scanning to measure real-world protocol deployment and to uncover flaws in cryptographic protocols [2], [14], [20], [21], [28], [30], [32]. However, for the most part, the security community has studied previously topical protocols, e.g., TLS and SSH. Similarly, there have been several recent papers that introduced tools for performing Internet-wide scans (e.g., ZMap [23]). We develop a new FTP enumerator but build on existing tools, using ZMap to perform host discovery. The work most related to FTP scanning is FTP Map, which attempts to detect FTP vulnerabilities based on server behavior [27]. Our goals differ significantly as we are trying to enumerate the file structure of FTP servers. There exists an online search engine for FTP servers, Napalm FTP Indexer [37], which contains files from 17K FTP servers. In comparison, our scan revealed 1.1M anonymous FTP servers.

Embedded Device Security While there is considerable anecdotal evidence of security issues plaguing embedded devices, there has been little systematic analysis of how these vulnerabilities affect real-world users. Our study investigates one protocol that has been commonly deployed by embedded devices, but other protocols likely suffer from similar issues. In a few recent examples, Durumeric et al. found that embedded devices did not patch in response to the OpenSSL Heartbleed bug [22], Heninger et al. found that embedded devices lacked the entropy needed to generate secure cryptographic keys [30], and Bokoski et al. found that SuperMicro devices suffered from elementary security vulnerabilities (e.g., trivial buffer overflows) and were publicly accessible [10].

Launching Attacks Previous research has explored the risk of devices interacting with the same resource through multiple channels. Bojinov introduced the concept of Cross Channel Scripting (XCS), in which an attacker injects malicious content into web content over a secondary protocol (e.g., SMB or FTP) [9]. Several of the malicious use cases we see (e.g., SEO and DDoS attacks) are instances of this where FTP is being used as the secondary communication channel.

XII. Conclusion

In this study, we presented a comprehensive analysis of how anonymous FTP has been deployed in the real world. We found that despite the protocol being largely forgotten by the research community, there are more than 13.8M FTP servers on the IPv4 address space, of which 1.1M (8%) allow anonymous access. Unfortunately, many anonymous FTP servers expose sensitive data, ranging from cryptographic secrets to confidential financial documents. We fingerprinted servers hosting this data and found that many are consumer devices (e.g., home NAS devices) that expose data over anonymous FTP by default or fail to explain the risks of enabling anonymous FTP. We further uncovered evidence that nearly 20K FTP servers allow anonymous users to write data, which malicious actors are using for malware deployment, and click fraud, and DDoS attacks. Our study presents a grim portrait of how FTP is deployed in 2015, but we hope that shedding light on the issue will prompt the security community to begin to address these vulnerabilities.

Acknowledgments

The authors wish to thank Mudge Zatko for helping motivate this analysis and providing insight into possible threats. We thank the exceptional IT staff at the University of Michigan for their help and support, including Chris Brenner, Kevin Cheek, Laura Fink, Dan Maletta, Jeff Richardson, Don Winsor, Donald Welch, and others from ITS, CAEN, and DCO. This material is based in part upon work supported by the U.S. National Science Foundation under grants CNS-1345254, CNS-1409505, and CNS-1518888, by the NSF Graduate Research Fellowship Program under grant DGE-1256260, by the Post-9/11 GI Bill, by the Google Ph.D. Fellowship in Computer Security, and by an Alfred P. Sloan Foundation Research Fellowship.

References

[1] http://www.bitwixen.com.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelin, and P. Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[3] M. Allman and S. Ostermann. FTP security considerations. RFC 2577, May 1999.

[4] ASUS. WL-700gE user manual, July 2006. http://dlcdnet.asus.com/pub/ASUS/wireless/WL-700g/WL700gE_manual_E2393_high.zip.

[5] ASUS. User guide RT-AC68R, Aug. 2013. http://dlcdnet.asus.com/pub/ASUS/wireless/RT-AC68R/E8617_RT_AC68R_Manual_English.pdf.

[6] S. Bellovin. Firewall-Friendly FTP. RFC 1579 (Informational), Feb. 1994.

[7] A. Bhushan. File Transfer Protocol. RFC 114, Apr. 1971. Updated by RFCs 133, 141, 171, 172.

[8] Bitmessage Wiki. https://bitmessage.org/wiki/Main_Page.

[9] H. Bojinov, E. Bursztein, and D. Boneh. XCS: Cross channel scripting and its impact on web applications. In *16th ACM Conference on Computer and Communications Security*, Oct. 2009.

[10] A. Bonkoski, R. Bielawski, and J. A. Halderman. Illuminating the security issues surrounding lights-out server management. In *8th USENIX Workshop on Offensive Technologies*, Aug. 2013.

[11] Buffalo. Buffalo Linkstation FTP setup guide. http://site2.buffalotech.com/downloads/FTP%20Setup%20Guide.pdf.

[12] Buffalo. Linkstation 400 user manual, June 2015. http://manual.buffalo.jp/buf-doc/35020812-02_EN.pdf.

[13] CERT. FTP Bounce. CERT Advisory CA-1997-27, Dec. 1997. https://www.cert.org/historical/advisories/ca-1997-27.cfm.

[14] S. Checkoway, M. Fredrikson, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, and H. Shacham. On the practical exploitability of Dual EC in TLS implementations. In *23rd USENIX Security Symposium*, Aug. 2014.

[15] CVE-2015-3306, Apr. 2015. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3306.

[16] Dark Tangent. Cyberspace Underwriters Laboratories, Jan. 1999. https://dl.packetstormsecurity.net/docs/infosec/cyberul.html.

[17] P. Deutsch, A. Emtage, and A. Marine. How to use Anonymous FTP. RFC 1635, May 1994.

[18] DSNS network scanner. http://www.dsns.net/.

[19] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by Internet-wide scanning. In *22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[20] Z. Durumeric, D. Adrian, A. Mirian, J. Kasten, E. Bursztein, N. Lidzborski, K. Thomas, V. Eranti, M. Bailey, and J. A. Halderman. Neither snow nor rain nor mitm: An empirical analysis of email delivery security. In *15th ACM Internet Measurement Conference*, pages 27–39. ACM, 2015.

[21] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *13th ACM Internet Measurement Conference*, Oct. 2013.

[22] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of Heartbleed. In *14th ACM Internet Measurement Conference*, Nov. 2014.

[23] Z. Durumeric, E. Wustrow, and J. A. Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *22nd USENIX Security Symposium*, Aug. 2013.

[24] Exploit4Arab. Seagate Central FTP root access. http://www.exploit4arab.net/exploits/1530.

[25] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving SSL warnings: Comprehension and adherence. In *33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2893–2902. ACM, 2015.

[26] P. Ford-Hutchinson. Securing FTP with TLS. RFC 4217, Oct. 2005.

[27] FTP-Map. https://github.com/Hypsurus/ftpmap.

[28] O. Gasser, R. Holz, and G. Carle. A deeper understanding of SSH: results from Internet-wide scans. In *2014 Symposium on Network and Distributed System Security*, Feb. 2014.

[29] Google. Robots.txt specification. https://developers.google.com/webmasters/control-crawl-index/docs/robots_txt.

[30] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *21st USENIX Security Symposium*, Aug. 2012.

[31] A. Klein. Filezilla FTP server is vulnerable to FTP PORT bounce attack and PASV connection theft. http://www.securitygalore.com/site3/filezilla_ftp_server_advisory.

[32] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *2015 Network and Distributed System Security Symposium*, Feb. 2015.

[33] LaCie. CloudBox user manual. http://manuals.lacie.com/EN/manuals/cb/01_intro/start.

[34] Linksys. Overview of the security tool in Linksys Smart Wi-Fi. http://www.linksys.com/us/support-article?articleNum=140559.

[35] Linksys. Quick USB storage setup guide for Linksys storage link routers. http://www.linksys.com/eg/support-article?articleNum=142291.

[36] N. Mathewson and N. Provos. LibEvent: An event notification library. http://libevent.org.

[37] Napalm FTP indexer. http://www.searchftps.net/.

[38] P. Paganini. ASUS routers setting could expose users' data on Internet. Security Affairs, Jan. 2014. http://securityaffairs.co/wordpress/21212/hacking/asus-routers-hack.html.

[39] Symantec. W32.Ramnit analysis, Feb. 2015. https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32-ramnit-analysis.pdf.

[40] Synology. Synology NAS user's guide. http://global.download.synology.com/download/Document/UserGuide/DSM/5.2/Syno_UsersGuide_NAServer_enu.pdf.