Security Assessment of Streisand for Open Technology Fund

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Scope and Methodology

IncludeSec performed a security assessment of Streisand for Open Technology Fund. The assessment team performed a 5 day effort spanning from March 19th – March 23, 2018. The scope of the assessment was limited to a security configuration review of the Streisand HTTP Gateway, VPN, Proxy services, ansible playbooks, and Streisand hosts for hardening recommendations.

Assessment Objectives

The objective of this assessment was to identify and confirm potential security vulnerabilities within the in-scope targets. The team assigned a qualitative risk ranking to each finding. IncludeSec also provided remediation steps which Streisand could implement to secure its applications and systems.

Findings Overview

IncludeSec identified 15 categories of findings. There were 0 deemed a “Critical-Risk,” 4 deemed a “High-Risk,” 3 deemed a “Medium-Risk,” and 8 deemed a “Low-Risk,” which pose some tangible security risk.

IncludeSec encourages Streisand to redefine the stated risk categorizations internally in a manner that incorporates internal knowledge regarding business model, customer risk, and mitigation environmental factors.

Next Steps

IncludeSec advises Streisand to remediate as many findings as possible in a prioritized manner and make systemic changes to the Software Development Life Cycle (SDLC) to prevent further vulnerabilities from being introduced into future release cycles. This report can be used by Streisand as a basis for any SDLC changes. IncludeSec welcomes the opportunity to assist Streisand in improving their SDLC in future engagements by providing security assessments of additional products.

ASSESSMENT RESULTS

At the conclusion of the assessment, Include Security categorized findings into four levels of perceived security risk: critical, high, medium, or low. Any informational findings for which the assessment team perceived no direct security risk, were also reported in the spirit of full disclosure. The risk categorizations below are guidelines that IncludeSec believes reflect best practices in the security industry and may differ from internal perceived risk. It is common and encouraged that all clients recategorize findings based on their internal business risk tolerances. All findings are described in detail within the final report provided to Streisand Effect.

Critical-Risk findings are those that pose an immediate and serious threat to the company's infrastructure and customers. This includes loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information. These threats should take priority during remediation efforts.

High-Risk findings are those that could pose serious threats including loss of system, access, or application control, compromise of administrative accounts or restriction of system functions, or the exposure of confidential information.

Medium-Risk findings are those that could potentially be used with other techniques to compromise accounts, data, or performance.

Low-Risk findings pose limited exposure to compromise or loss of data, and are typically attributed to configuration issues, and outdated patches or policies.

Informational findings pose little to no security exposure to compromise or loss of data which cover defense-in-depth and best-practice changes which we recommend are made to the application.

The findings below are listed by a risk rated short name (e.g., C1, H2, M3, L4, I5) and finding title. Each finding includes: Description (including proof of concept screenshots and lines of code), Recommended Remediation, and References.

INTRODUCTION

The assessment team performed an audit of Streisand, an open source product that sets up VPN and Proxy services according to the user's preference either locally or in the cloud. The conducted assessment was time-boxed to 5 days.

The scope of the assessment was limited to security configuration review of the Streisand HTTP Gateway, VPN, Proxy services, Ansible playbooks, and Streisand hosts for hardening recommendations. Hardening recommendations were selected to protect communications which are considered unclassified. The Streisand project doesn't guarantee anonymity as a first-class property; thus, anonymity was not a primary focus for the assessment.

Supported Services

VPN & Proxy Services

The VPN and Proxy services supported by Streisand are:

L2TP/IPsec - IPSec was configured to use a pre-shared key for authentication. Disclosure of the pre-shared key may allow an adversary to man-in-the-middle communications. Additionally, cipher suites were found to support 3DES, lacked perfect-forward-secrecy, and made use of the weak Diffie-Hellman MODP1024 group. As a result, adversaries may be able to decrypt communications.

OpenConnect – Cipher suites involving 3DES and lacking perfect forward secrecy were used. As a result, adversaries may be able to decrypt communications. Additionally, IPv6 traffic did not route through the VPN service, increasing the risk of Streisand users being censored.

OpenVPN - IPv6 traffic did not route through the VPN service, increasing the risk of Streisand users being censored.

OpenSSH Tunneling - Supported cipher suites are acceptable for communications which are not TOP SECRET. Arbitrary commands were not able to executed as the **forward** user, due to the user's **authorized_keys** file specifying a default **echo** command. X11 forwarding was also disabled. When Streisand was deployed to AWS, the deployment **ubuntu** account was still accessible through OpenSSH. This account could be used to gain root privileges on the Streisand host if the machine which deployed the instance were ever compromised. As SSHing into a Streisand instance with this user for administrative purposes does not appear to be standard practice, consider removing the SSH key used by Ansible once the instance is live as defense-in-depth measure.

Shadowsocks - The Shadowsocks protocol did not support perfect forward secrecy. Additionally, Shadowsocks makes use of a pre-shared key for authentication, which may allow

an attacker to man-in-the-middle connections if the key were ever obtained by an attacker. Per Streisand's documentation on line 64 of **README.md**, sharing of the pre-shared key with the public through QR codes is suggested. The assessment team recommends keeping pre-shared key private. Because of the nature of the key, only sharing it with trusted individuals who will use the proxy is recommended.

Stunnel - The use of anonymous cipher suites was discovered which may allow an adversary to man-in-the-middle communications.

Tor - Support for JavaScript and multimedia is enabled by default. This broadens the attack vector and provides adversaries more targets to exploit to compromise a system.

Wireguard - No configuration issues were noted during the assessment.

Miscellaneous Services

The Streisand application ships with monit, a daemon which was used to auto-restart applications if closed unexpectedly.

Streisand enables most of the services described above by default. The following code snippet lists the services enabled by default for a local deployment of Streisand. Configurations for third party providers is similar.

From **streisand/global_vars/noninteractive/local-site.yml**:

```
16 streisand_l2tp_enabled: yes
17 streisand_openconnect_enabled: yes
18 streisand_openvpn_enabled: yes
19 streisand_shadowsocks_enabled: yes
20 streisand_ssh_forward_enabled: yes
21 # By default sshuttle is disabled because it creates a `sshuttle` user that has
22 # full shell privileges on the Streisand host
23 streisand_sshuttle_enabled: no
24 streisand_stunnel_enabled: yes
25 streisand_tinyproxy_enabled: yes
26 streisand_tor_enabled: yes
27 streisand_wireguard_enabled: yes
```

To limit the attack surface area, the assessment team recommends reducing the number of services enabled by default or encouraging users to disable any unused services.

General Security Recommendations

Provide a Secure Browser Environment

When Streisand Proxies were used, IP addresses were found to be disclosed through the use of WebRTC in web browsers. Disable WebRTC and browser extensions for a stronger security posture. Additionally, users may be tracked as browser history and cookies are not cleared. Providing a live-distro which contains a hardened environment is recommended.

Credential Generation & Storage

Some services provided by Streisand rely on pre-shared keys, which may result in communications being vulnerable to a man-in-the-middle attack. Whenever possible, utilize public key cryptography for authentication. Additionally, credentials to connect to VPN proxies and clients were stored in plaintext on disk. Design the service to encrypt sensitive credentials whenever possible to mitigate the risk of disclosure. For more details and suggestions, review the Appendix **Credential Complexity, Usage, and Storage Considerations**.

Third Party Host Providers

Third-party host providers cannot necessarily be considered trustworthy, as these providers have full control over the hosted computing device. Due to recent Meltdown and Spectre vulnerabilities, operating Streisand services on unpatched shared computing environments may result in the unintended disclosure of private keys.

Brief Users on Security Operations

Let users know that using cryptographic algorithms and protocols is not enough to secure communications or conceal a user's identity. Passwords, keys, and communications are often compromised when stored or transferred insecurely, such as in plaintext format. Exploiting software vulnerabilities may also be used to gain access to systems. Suggest that software is regularly updated on the users' machines, as well on Streisand's servers. Encourage users to always use end-to-end encryption to protect communications.

Monitor Software for Updates

The **monit** package for Ubuntu Xenial Ubuntu 16.04.4 LTS (Xenial Xerus) is out of date and missing a backport patch for a cross-site request forgery vulnerability. Software that Streisand depends upon should be monitored for updates to ensure security patches are always available. Using a more up-to-date version of Ubuntu, such as a regular release build, may mitigate the risk of outdated software being used.

The assessment team recommends adding a mechanism to automatically check for software updates and provide a simple method for users to update outdated software. This could be accomplished by creating a separate application which is developed and distributed with Streisand for instance.

Ensure VPN Clients Can Not Communicate With Each Other

Due to time limitations, client-to-client communications was not assessed. To mitigate the risk of client machines being attacked by users on the same VPN, ensure clients cannot send network traffic to one another.

Disable Access to Provisioning Linux Account

The account **ubuntu** was able to be logged into remotely through the use of public key authentication. The account can also obtain root privileges through the use of **sudo**. Remote access to the provisioning account through SSH may be revoked to further harden the system.

HIGH-RISK FINDINGS

H1: [OpenVPN,OpenConnect] IPv6 Traffic Disclosed

Description:

The Streisand application did not configure OpenVPN and OpenConnect to handle IPv6 traffic. This may lead to Streisand users being censored or identified.

Many operating systems support a dual-stack configuration for IPv4 and IPv6. The Streisand application configured OpenVPN and OpenConnect to only handle IPv4 traffic. As the IPv6 network stack was not configured by Streisand, IPv6 network traffic did not flow through the VPN connection.

Shown below are the configuration lines for **OpenConnect** address leases. Note the **ipv4-network** directive was specified to configure leasing of IPv4 address. The **ipv6-network** was not specified, thus IPv6 addresses were not leased.

From **src/streisand/playbooks/roles/openconnect/templates/config.j2**:

```
39 ipv4-network = 192.168.1.0
40 ipv4-netmask = 255.255.255.0
```

For OpenConnect's server configuration, note the **server-ipv6** directive was not specified:

From **src/streisand/playbooks/roles/openvpn/templates/etc_openvpn_server.conf.j2**:

```
1 server 10.8.0.0 255.255.255.0
2 push "dhcp-option DNS {{ dnsmasq_openvpn_tcp_ip }}"
3 proto tcp
4 port {{ openvpn_port }}
5 {% include "etc_openvpn_server_common.j2" %}
```

Due to time constraints, only the Streisand clients for OpenVPN, OpenConnect, and TOR on macOS Sierra were reviewed for IPv6 leaks.

Proof of Concept

1. The website https://ipleak.net was viewed to obtain the client's public **IPv4** and **IPv6** address.

2. A connection to the **OpenVPN** host was created by running the command:

```
sudo openconnect --cafile ca.crt --certificate goat-mad.p12 --key-password
'resource.autumn.identify' 54.241.223.91:4443
```

Note in the **openconnect** output, an IPv4 address was only leased by the **OpenConnect** server daemon.

```
Connected as 192.168.1.218, using SSL
Established DTLS connection (using GnuTLS). Ciphersuite (DTLS1.2)-(PSK)-(3DES-CBC)-(SHA1).
add host 54.241.223.91: gateway 192.168.0.1
add net 192.168.1.0: gateway 192.168.1.218
delete net default: gateway 192.168.0.1
add net default: gateway 192.168.1.218
```

3. After connecting to the VPN, the website https://ipleak.net was viewed to obtain the client's **IPv4** and **IPv6** address again. The reported **IPv4** address was different from step one, meaning traffic was routed through the VPN. However, the **IPv6** address reported did not differ from step one, meaning the traffic did not route through the VPN.

Recommended Remediation:

The assessment team recommends routing IPv6 traffic through the VPN. If the host does not support **IPv6** but the client does, disable the client's IPv6 interface while connected to the VPN.

References:

OpenConnect Server ipv6-network Directive Example
IPv6 in OpenVPN

H2: [Monit] Out-of-Date Components in Use

Description:

The assessment team found the **monit** application contained a missing backported security patch. This may have allowed attackers to disable VPN and Proxy services on the Streisand host by exploiting a Cross-Site Request Forgery vulnerability within monit.

The severity of the finding has been marked as **High** risk, as Streisand relies on Ubuntu's package maintainers for software updates and security fixes, and this repository did not contain a backported security patch for monit. As nobody can predict the severity or exploitability of future vulnerabilities discovered within Streisand dependencies or guarantee that these will be patched by Ubuntu package maintainers, the assessment team has defaulted to ranking this issue as a **High** risk; a case could be made for ranking this at any risk level between Critical or Informational.

The monit application, version 5.16, was installed and out-of-date. This version contained a known Cross-Site Request Forgery vulnerability. Note that monit was installed using **apt**, which

used Ubuntu's package repository for Xenial (16:04 LTS). The Ubuntu repository package for monit contained the outdated version. Although the Cross-Site Request Forgery vulnerability was fixed in a future monit release, the patch was not backported to Ubuntu's Xenial repository. When checked, the latest monit version available was 5.25.1.

The following table are software packages that were found to be forked from their original software repository. All version numbers were obtained by running the applications through the command line interface and inspecting the program's output.

Component	Version in Use	Version Available	Installed By	Notes
monit	5.16	5.25.1	apt-get	**Vulnerable to Cross Site Request Forgery (CVE-2016-7067)**
dnsmasq	2.75	2.79	apt-get	
xl2tpd	1.3.6	1.3.11	apt-get	
stunnel	5.30	5.45	apt-get	
tinyproxy	1.8.3	1.8.4	apt-get	
openssl	1.0.2g	1.0.2o	Part of Ubuntu distribution	

Packages other than monit listed above do not necessarily contain known security vulnerabilities. Due to time constraints, creation or application of all backported security patches was not investigated. Thus, additional backported security patches may be missing and should be investigated.

Recommended Remediation:

The assessment team recommends updating all out-of-date components to their most recent releases. Software which is not kept in sync with the original source code repository must be continually reviewed to ensure existing and new security backports are being applied, when applicable. Components which have reach their end-of-life must not be used. Warn Streisand users of any known limitations or issues regarding updates to third-party libraries.

References:

OWASP A9 – Using Components With Known Vulnerabilities
Ubuntu monit package details for Xenial
Monit Source Code Repository

H3: [IPSec] Deprecated Cipher Suites and Authentication Methods Supported

Description:

Streisand's IPSec configuration used deprecated cipher suites. The cipher suites were used to provide network-layer security to ensure data confidentiality, authenticity, and integrity. If weak algorithms are not disabled, communications may be able to be decrypted. Additionally, IPsec was configured to use a pre-shared key. An attacker who gains access to the pre-shared may be able to perform a man-in-the-middle attack to decrypt communications.

The Streisand IPSec configuration allowed the use of cipher suites involving 3DES. The 3DES cipher is not secure for encrypting communications and should not be used for transport security. A cryptographic attack known as SWEET32 demonstrated how plaintext can be recovered from ciphertext encrypted by 64-bit block ciphers like 3DES. Additionally, cryptographic vulnerabilities against 3DES were found to reduce the cipher's effective key size. NIST urges all users of 3DES to migrate to AES as soon as possible.

Shown below is the configuration where 3DES was supported as a cipher suite.

From **src/streisand/playbooks/roles/l2tp-ipsec/templates/ipsec.conf.j2**:

```
26    ike=3des-sha1,3des-sha1;modp1024,aes-sha1,aes-sha1;modp1024,aes-sha2,aes-
sha2;modp1024,aes256-sha2_512
27    phase2alg=3des-sha1,aes-sha1,aes-sha2,aes256-sha2_512
```

The Streisand IPSec configuration uses the Diffie-Hellman MODP 1024-bit group for key exchanges. Diffie-Hellman MODP groups consisting of 1024 bits or less are vulnerable to a pre-computation attack which drastically reduces the time required to break individual session keys based on Diffie-Hellman Group. It is believed that nation states can break Diffie-Hellman key exchanges which use the MODP1024 bit group, allowing for communications to be eavesdropped upon.

Shown below is the configuration where the Diffie-Hellman group MOPD1024 is used.

From **src/streisand/playbooks/roles/l2tp-ipsec/templates/ipsec.conf.j2**:

```
26    ike=3des-sha1,3des-sha1;modp1024,aes-sha1,aes-sha1;modp1024,aes-sha2,aes-
sha2;modp1024,aes256-sha2_512
```

The IPSec configuration was also found to use a pre-shared key. An attacker who discovers the pre-shared key can man-in-the-middle communications. Additionally, perfect forward secrecy was not enabled. Multiple sessions may use the same encryption key, allowing communications of multiple session to be decrypted if the key is recovered.

Shown below is the configuration where pre-shared keys for authentication is enabled (**authby**) and perfect forward secrecy is disabled (**pfs**).

From **src/streisand/playbooks/roles/l2tp-ipsec/templates/ipsec.conf.j2**:

```
22    authby=secret
23    pfs=no
```

The IPSec configuration enables the use of the draft RFC HMAC-SHA-256-96, which truncates the output of HMAC-SHA-256 to 96 bits. Based on libreswan's documentation, there are devices which use HMAC-SHA-256-96 when securing IPSec traffic, such as Android 6.0 devices. Enabling this feature may reduce compatibility with newer devices, as newer devices expect a non-truncated HMAC-SHA-256 value.

The following line enables the use of HMAC-SHA-256-96:

From **src/streisand/playbooks/roles/l2tp-ipsec/templates/ipsec.conf.j2**:

```
28    sha2-truncbug=yes
```

Recommended Remediation:

The assessment team recommends disabling insecure cipher suites. Use a Diffie-Hellman MODP group of at least 2048 bits, or a Diffie-Hellman ECP group of 256 bits. Do not use 3DES to encrypt communications. Additionally, disable support for **MD5** and **SHA-1**, as libreswan will be dropping support. Avoid the use of pre-shared symmetric keys whenever possible. Use certificate-based authentication or authentication with RSA and ECDSA keys instead. Enable perfect forward secrecy to better secure communications. Disable the **sha2-truncbug** setting. Use HMAC-SHA-512 to avoid IPSec compatibility issues if required.

References:

libreswan – IPSec configuration and connections
Strong Swan – Security Considerations
Weak Diffie-Hellman and the Logjam Attack
Logjam: Diffie-Hellman, discrete logs, the NSA, and you
Sweet32: Birthday attacks on 64-bit block ciphers in TLS and OpenVPN
Update to Current Use and Deprecation of TDEA
Additional Diffie-Hellman Groups for Use with IETF Standards
libreswan – Using SHA2_256 for ESP connection establishes but no traffic passes (especially Android 6.0)
Draft RFC – The HMAC-SHA-256-96 Algorithm and Its Use With IPsec

H4: [Gateway] Installation of Self Signed Trusted Root Certification Authority Certificate

Description:

The Streisand Gateway instructions had the user add a self-signed Trusted Root Certification Authority certificate to their system. An attacker who obtains the private key used to sign the certificate may be able to man-in-the-middle SSL and TLS connections between the user and arbitrary hosts.

Note that the attack window is shortened by the fact that the Root CA private key is deleted immediately after it's used to sign nginx's self-signed certificate on the remote Streisand instance. Because the potential impact of this issue is significant, however, and there are still a number of scenarios where the private key for this CA could be disclosed or subverted, the assessment team considers this a High-risk issue.

When connecting to a service over SSL or TLS, X.509 certificates are verified by using a chain of trust. At the top of the certificate chain is the Root Certificate Authority certificate, which is used for verifying the authenticity of undersigned certificates. If an attacker compromises the private key used by a Certificate Authority to sign certificates, the attacker may create and sign certificate as the Certificate Authority. These certificates will be trusted by SSL and TLS supported applications allowing the connections to be man-in-the-middled, assuming certificate pinning is not in use and the application accepts certificates signed by the Certificate Authority. As it's critical to protect the private key for root certificates, it is discouraged for applications to suggest or add root certificates to the system.

Additionally, Certificate Authorities can revoke certificates before they expire, such as through the use of the Online Certificate Status Protocol. Revocation of a certificate may be used when the private key used to sign a certificate is compromised. As a self-signed certificate was used, no process exists to automate the revocation of the certificate.

Attack Scenarios

The private key used to sign the Root CA Certificate is generated on the remote Streisand instance. After the Root CA private key is used to sign nginx's self-signed certificate, the root CA private key is automatically deleted.

Although the window of time where the private key is available on the system is small, there are a number of potential attack scenarios that would have a significant impact on users of Streisand. These include:

1. The 3rd party hosting service (e.g. AWS's Ubuntu installation) contains a backdoor, allowing the private key to be recovered.

2. A request from a nation state to image the system is honored, ultimately allowing the nation state to intercept user traffic.
3. An attacker obtains root access to the system, possibly recovering the private key by reading directly from the disk device or dumping memory (since Ansible's **file** module was used to delete the material).

Another potential concern is that an untrustworthy Streisand administrator could create their own CA and certificate file and update the instructions that will be shared with other users to include this certificate, allowing them to MitM TLS connections between Streisand users and the instance, given a suitable network position.

The following shows where the self-signed certificate is recommended to be added as a root certificate.

From **streisand/playbooks/roles/streisand-gateway/templates/instructions.md.j2**:

```
43 These instructions work for Chrome and Internet Explorer. Firefox uses its own internal
Certificate Manager, but it's [easy to configure](#ssl-firefox) too.
44
45 1. [Download the SSL certificate](#download) embedded in this document.
46 1. Double-click to open the downloaded certificate.
47 1. Click *Install Certificate...*
48 1. The Certificate Import Wizard will start. Click *Next*.
49 1. Select *Place all certificates in the following store* and click *Browse*.
50 1. Select *Trusted Root Certification Authorities* and click *OK*.
51 1. Click *Next* and then click *Finish*.
52 1. Confirm that you would like to install the certificate by choosing *Yes*.
53 1. Close and re-open your browser.
54 1. You are ready to connect. See [Connecting to your Streisand Gateway](#connecting-ssl).
```

Note the above instruction template file has additional instructions for macOS, iOS, Firefox, Chromium, and Android users.

The Ansible playbook instructions used to create and delete the private key are below. From **streisand/playbooks/roles/streisand-gateway/tasks/openssl.yml**:

```
18 - name: Create private key and self-signed certificate for Nginx Certificate Authority.
19   command:
20     openssl req -new -nodes -x509
21     -config "{{ openssl_ca_base }}/openssl-local.cnf"
22     -days {{ nginx_days_valid }}
23     -extensions v3_ca
24     -newkey rsa:{{ streisand_gateway_rsa_key_size }}
25     -keyout {{ openssl_ca_key }}
26     -out {{ openssl_ca_certificate }}
27     -subj "/C={{ nginx_key_country }}/ST={{ nginx_key_province }}/L={{ nginx_key_city
}}/O={{ nginx_key_org }}/OU={{ nginx_key_ou }}/CN=Streisand {{ streisand_ipv4_address }} Root
CA"
28   args:
29     creates: "{{ openssl_ca_certificate }}"
```

```
...
69 - name: Remove the CA private key. It has signed its first and last certificate.
70   file:
71     path: "{{ openssl_ca_key }}"
72     state: absent
```

Proof of Concept

The following steps demonstrate how adding a self-signed certificate to the system allows connections to https://example.com to be man-in-the-middled.

1. Obtain and install **sslsplit** from https://github.com/droe/sslsplit.
2. Generate a X.509 Certificate using **openssl**.

```
cd /tmp
openssl genrsa -out ca.key 4096
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
```

3. Start **sslsplit** to proxy **localhost:3229** to **example.com:443**.

```
sslsplit -k ca.key -c ca.crt https localhost 3249 www.example.com 443
```

4. In Google Chrome, navigate to https://localhost:3249. Note the browser displays a certificate error, preventing the user from connecting to the site.
5. Add the certificate created in step three as a Trusted Root Certificate Authority Certificate to the system.
6. Refresh https://localhost:2349. Note how the browser does not display a certificate error.
7. Remove the trusted root certificated added in step five.

Recommended Remediation:

The assessment team recommends not requiring users to add a self-signed Trusted Root Certification Authorities certificate to their systems. If gateway instructions must be shared with other users, update documentation on how to share the instructions securely, such as by using PGP. If mirror content must be provided over HTTP, ensure instructions are added on how to verify downloads, such as by verifying signatures of checksums on the files. Ensure users uninstall self-signed root certificates provided by Streisand.

References:

Malware Bytes – When You Shouldn't Trust a Trusted Root Certificate

MEDIUM-RISK FINDINGS

M1: [Gateway] Deprecated TLS Ciphers Supported

Description:

The application's HTTPS service uses deprecated algorithms or protocols. The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are the encryption components of the HTTPS protocol suite. They provide transport-layer security to ensure data confidentiality, authenticity, and integrity. If weak algorithms are not disabled, a suitably located attacker could mount a cipher downgrade attack, forcing both the client and the server to use a cipher weaker than the one they would have normally chosen.

The deprecated protocol TLS 1.0 was found to be in use. Notably, the protocol contains a vulnerability which allows plaintext to be recovered with CBC ciphers are used, under certain conditions. The BEAST attack demonstrated how a user's authentication tokens can be obtained by inspecting encrypted traffic, while forcing a user to perform multiple HTTPS requests. The vulnerability has been mitigated in browsers since 2011. The underlying protocol is still flawed however.

Several cipher suites were discovered to not support perfect forward secrecy. Perfect forward secrecy allows previous TLS connections to be secure if the host's private key is compromised. The Ephemeral Diffie-Hellman Key Exchange algorithm is commonly used to achieve perfect forward secrecy.

TLS cipher suites which use 3DES are considered insecure and were used by the application. The cryptographic SWEET32 attack affects ciphers which use a block size of 64 bits, such as 3DES, allowing plaintext to be recovered after 2^32 blocks with the same key have been encrypted. Additionally, cryptographic vulnerabilities of 3DES were found that reduces the cipher's effective key size.

Streisand was deployed as an EC2 instance. The following findings were discovered while reviewing SSL and TLS configurations for the Gateway application.

Protocol	Cipher Suite	SSL/TLS Protocol Version Not NIST Compliant For Government Systems	Lacks Perfect Forward Secrecy (DHE /ECDHE)	Weak Key Size (Less Than 128 Bits)	Additional Weaknesses with Cipher Suite or Protocol
TLS 1	ECDHE-RSA-AES256-SHA (TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA)	X			**X**

TLS 1	DHE-RSA-AES256-SHA (TLS_DHE_RSA_WITH_AES_256_CBC_SHA)	X			**X**
TLS 1	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)	X	X		**X**
TLS 1	ECDHE-RSA-AES128-SHA (TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA)	X			**X**
TLS 1	DHE-RSA-AES128-SHA (TLS_DHE_RSA_WITH_AES_128_CBC_SHA)	X			**X**
TLS 1	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)	X	X		**X**
TLS 1	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)	X		112	**X**
TLS 1	EDH-RSA-DES-CBC3-SHA (TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA)	X		112	**X**
TLS 1	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)	X	X	112	**X**
TLS 1.1	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)		X		
TLS 1.1	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)		X		
TLS 1.1	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.1	EDH-RSA-DES-CBC3-SHA (TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.1	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)		X	112	**X**
TLS 1.2	AES256-GCM-SHA384 (TLS_RSA_WITH_AES_256_GCM_SHA384)		X		
TLS 1.2	AES256-SHA256 (TLS_RSA_WITH_AES_256_CBC_SHA256)		X		
TLS 1.2	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)		X		

TLS 1.2	AES128-GCM-SHA256 (TLS_RSA_WITH_AES_128_GCM_SHA256)		X		
TLS 1.2	AES128-SHA256 (TLS_RSA_WITH_AES_128_CBC_SHA256)		X		
TLS 1.2	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)		X		
TLS 1.2	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.2	EDH-RSA-DES-CBC3-SHA (TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.2	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)		X	112	**X**

The following excerpt shows the nginx configuration template which enabled the above cipher suites.

From **streisand/playbooks/roles/streisand-gateway/templates/vhost.j2**:

```
29   ssl_ciphers 'ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:ECDHE-ECDSA-
AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-AES128-
SHA256:ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-
AES128-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA:ECDHE-RSA-AES256-SHA:DHE-RSA-
AES128-SHA256:DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES256-SHA:ECDHE-ECDSA-DES-
CBC3-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES128-GCM-SHA256:AES256-GCM-
SHA384:AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:DES-CBC3-SHA:!DSS';
```

Recommended Remediation:

The assessment team recommends reconfiguring the HTTPS service to ensure that attempts to anything but the latest TLS version are rejected. Reconfigure cipher suites to ensure only secure configurations are used. The application **testssl.sh** may be used to aid in TLS configuration review.

References:

Testing for SSL-TLS (OWASP-CM-001) – OWASP
Are You Ready for 30 June 2018? Saying Goodbye to SSL/early TLS
NIST Special Publication 800-52 Revision 1 – Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
Development and Implementation of Secure Web Applications

Sweet32: Birthday attacks on 64-bit block ciphers in TLS and OpenVPN
Here Come The XOR Ninjas
Lucky Thirteen: Breaking the TLS and DTLS Record Protocols

M2: [Monit] Sensitive Operations Vulnerable to Cross-Site Request Forgery

Description:

The **monit** application, which is exposed through the Streisand Gateway, is vulnerable to cross-site request forgery (CSRF). In a CSRF attack, an attacker forces an application to perform an action on behalf of a targeted user. This is accomplished by tricking the user's browser into performing a state-changing request to the application while the user is authenticated with the system. Since the user is authenticated, the action is performed in the context of the user's session.

A request that carries out a function (such as disabling Proxy daemons) based on data in a GET or POST request is vulnerable to CSRF attacks if it is used to perform that function in a single stage with no validation or assurance that the request originated from an interactive browser session. Note that CSRF attacks can also be performed against applications that are not accessible from the Internet, such as intranet sites or device management interfaces like a router's administration console. The key in these scenarios would be that the targeted user's browser is able to reach those systems because they are accessible from the user's workstation.

None of the actions in the **monit** application required a secure random token (also called an anti-CSRF token) or other value to validate requests and prevent attackers from performing CSRF attacks.

Proof of Concept

The following HTML document was used to perform a CSRF attack to disable the **sslh** service on the Streisand instance **54.241.223.91**.

```
<html>
<form method="POST" action="https://54.241.223.91/monit/sslh"/>
<input name="action" value="stop">
<input type="submit">
</form>
</html>
```

When an authenticated user to the **Streisand Gateway** loads the above document in their browser, it will submit the embedded form, disabling the **sslh** service. The CSRF vulnerability is in the **monit** dependency, which is outdated

The following shows Streisand Gateway's nginx configuration which exposed the **monit** web server through the URI **https://streisand-gateway/monit**.

From **streisand/playbooks/roles/streisand-gateway/templates/vhost.j2**:

```
52   location /monit/ {
53     rewrite ^/monit/(.*) /$1 break;
54     proxy_ignore_client_abort on;
55     proxy_pass   http://localhost:2812;
56   }
```

Recommended Remediation:

The assessment team recommends updating the **monit** service to the latest version. Do not expose the **monit** web daemon through **nginx**.

References:

OWASP 2013-A8-Cross-Site Request Forgery (CSRF)

M3: [Shadowsocks] Known Protocol Flaws

Description:

Based on tickets opened in the Shadowsocks project issue tracker, there are known flaws with the Shadowsocks protocol. Knowledge of the shared private key may lead to connections being hijacked or communications being decrypted.

The use of forward secrecy has become a best practice when security communications. Essentially, forward secrecy ensures each communication session has a unique symmetric key, which must be attacked directly to decrypt or intercept the traffic. Based on a previously filed ticket, the Shadowsocks protocol does not appear to have this property.

Additionally, Shadowsocks makes uses of a symmetric pre-shared key, which must be shared with other clients wishing to connect to the proxy. An open ticket exists for Shadowsocks where anyone who knows the shared key and is in a position to man-in-the-middle traffic can hijack communications and decrypt traffic.

Note the Streisand **README.md** file suggests sharing instructions and QR codes to connect to Shadowsocks proxies is acceptable (line 64). The QR code contains the symmetric pre-shared key is used to secure traffic. As the code needs to be protected, sharing the QR code with untrusted individuals will allow the security to be compromised more easily. Furthermore,

attackers may distribute their own QR codes publicly, and have targeted individuals connect. This can lead to users being censored or spied on.

Due to project scoping and time constraints, the protocol was not reviewed for weaknesses.

Recommended Remediation:

The assessment team recommends to not suggesting sharing or distributing Shadowsocks QR codes with the public.

References:

Shadowsocks – Perfect Secrecy Missing
Shadowsocks – Man-in-the-middle protocol vulnerability
Shadowsocks – Proposed Threat Model

LOW-RISK FINDINGS

L1: [OpenConnect] Deprecated TLS and DTLS Ciphers Supported

Description:

The application's HTTPS service uses deprecated algorithms or protocols. The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are the encryption components of the HTTPS protocol suite. They provide transport-layer security to ensure data confidentiality, authenticity, and integrity. If weak algorithms are not disabled, a suitably located attacker could mount a cipher downgrade attack, forcing both the client and the server to use a cipher weaker than the one they would have normally chosen.

The deprecated protocol TLS 1.0 was found to be in use. Notably, the protocol contains a vulnerability which allows plaintext to be recovered with CBC ciphers are used, under certain conditions. The BEAST attack demonstrated how user's authentication tokens can be obtained by inspecting encrypted traffic, while forcing a user to perform multiple HTTPS requests. The vulnerability has been mitigated in browsers since 2011. The underlying protocol is still flawed however.

Several cipher suites were discovered to not support perfect forward secrecy. Perfect forward secrecy allows previous TLS connections to be secure if the host's private key is compromised. The Ephemeral Diffie-Hellman Key Exchange algorithm is commonly used to achieve perfect forward secrecy.

TLS cipher suites which use 3DES are considered insecure and were used by the application. The cryptographic SWEET32 attack affects ciphers which use a block size of 64 bits, such as 3DES, allowing plaintext to be recovered after 2^32 blocks with the same key have been encrypted. Additionally, cryptographic vulnerabilities of 3DES were found that reduces the cipher's effective key size.

The following TLS protocols and cipher suites and protocols were enumerated on the Streisand instance host and port **54.241.223.91:4443** using **testssl.sh**.

Protocol	Cipher Suite	SSL/TLS Protocol Version Not NIST Compliant For Government Systems	Lacks Perfect Forward Secrecy (DHE/ECDHE)	Weak Key Size (Less Than 128 Bits)	Additional Weaknesses with Cipher Suite or Protocol
TLS 1	ECDHE-RSA-AES256-SHA (TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA)	X			**X**

TLS 1	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)	X	X		**X**
TLS 1	CAMELLIA256-SHA (TLS_RSA_WITH_CAMELLIA_256_CBC_SHA)	X	X		**X**
TLS 1	ECDHE-RSA-AES128-SHA (TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA)	X			**X**
TLS 1	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)	X	X		**X**
TLS 1	CAMELLIA128-SHA (TLS_RSA_WITH_CAMELLIA_128_CBC_SHA)	X	X		**X**
TLS 1	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)	X		112	**X**
TLS 1	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)	X	X	112	**X**
TLS 1.1	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)		X		
TLS 1.1	CAMELLIA256-SHA (TLS_RSA_WITH_CAMELLIA_256_CBC_SHA)		X		
TLS 1.1	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)		X		
TLS 1.1	CAMELLIA128-SHA (TLS_RSA_WITH_CAMELLIA_128_CBC_SHA)		X		
TLS 1.1	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.1	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)		X	112	**X**
TLS 1.2	AES256-GCM-SHA384 (TLS_RSA_WITH_AES_256_GCM_SHA384)		X		
TLS 1.2	AES256-SHA256 (TLS_RSA_WITH_AES_256_CBC_SHA256)		X		
TLS 1.2	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)		X		

TLS 1.2	CAMELLIA256-SHA (TLS_RSA_WITH_CAMELLIA_256_CBC_SHA)		X		
TLS 1.2	AES128-GCM-SHA256 (TLS_RSA_WITH_AES_128_GCM_SHA256)		X		
TLS 1.2	AES128-SHA256 (TLS_RSA_WITH_AES_128_CBC_SHA256)		X		
TLS 1.2	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)		X		
TLS 1.2	CAMELLIA128-SHA (TLS_RSA_WITH_CAMELLIA_128_CBC_SHA)		X		
TLS 1.2	ECDHE-RSA-DES-CBC3-SHA (TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA)			112	**X**
TLS 1.2	DES-CBC3-SHA (TLS_RSA_WITH_3DES_EDE_CBC_SHA)		X	112	**X**

Additionally, 3DES ciphers were supported over DTLS on UDP port 4443. 3DES Cipher support for DTLS was tested by running the following command.

```
sudo openconnect --cafile ca.crt --certificate goat-mad.p12 --key-password
'resource.autumn.identify'  54.241.223.91:4443  --dtls-ciphers=DES-CBC3-SHA
```

The output of the command displayed the following line, indicating RSA-3DES-CBC-SHA1 was used as the ciphersuite.

```
Established DTLS connection (using GnuTLS). Ciphersuite (DTLS0.9)-(RSA)-(3DES-CBC)-(SHA1).
```

Cipher suites supported by DTLS and TLS are configured through the line below:

From **streisand/playbooks/roles/openconnect/templates/config.j2**:

```
25 tls-priorities = "NORMAL:%SERVER_PRECEDENCE:%COMPAT:-VERS-SSL3.0"
```

Note that Streisand supplies instructions to the user to use the **--pfs** argument with OpenConnect. This should prevent the OpenConnect client from using cipher suites which do not support perfect forward secrecy.

The **OpenConnect** instructions provided by Streisand can be seen below.

From **streisand/playbooks/roles/openconnect/templates/instructions.md.j2**:

```
67 1. Run OpenConnect:
68
69   `sudo openconnect --cafile ca.crt --certificate your-client-certificate.p12 --key-
password 'your-client-certificate-password' --pfs {{ streisand_ipv4_address }}:{{ ocserv_port
}}`
```

Recommended Remediation:

The assessment team recommends reconfiguring the HTTPS service to ensure that attempts to anything but the latest TLS version are rejected.

References:

Testing for SSL-TLS (OWASP-CM-001) – OWASP
PCI Compliance – Disable SSLv2 and Weak Ciphers (contains Apache and IIS instructions)
Development and Implementation of Secure Web Applications
testssl.sh

L2: [Ansible] Obtaining Deployed SSH Public Keys May Reveal Users' Identities

Description:

The Streisand application may upload a user's default SSH keys to remote hosts. If the remote host was to be breached or audited, and the public key were to be obtained, the identity of the user may be revealed.

OpenSSH uses the file path **~/.ssh/id_rsa.pub** for location for a user's SSH public key. Often, public keys contain a comment which include a user's real name, email address, or the account and hostname the key was generated on. By inspecting the contents of the public key, the identity of the user may be revealed. By default, the Streisand application uploads the default SSH public key to third-party providers, such as AWS, when provisioning a Streisand server instance.

Note even if a public key does not contain any personal identifiable information, a user's real identity may still be able to be revealed unintentionally. Other third-party services may obtain a user's default SSH public key and associate the key with the user's account. The user's account may then contain personal identifiable information, such as billing information. This would indirectly relate a user's SSH public key to their real identity. Furthermore, the public key may be published and searchable by various third parties. For instance, user's OpenSSH keys are publicly viewable on GitHub by navigating to https://github.com/<username>.keys, such as https://github.com/includesecurity.keys.

The following code shows the **streisand_ssh_private_key** variable set to the default file path of a user's SSH private key. Throughout Streisand, the **.pub** extension is added to the file path to obtain the public key path. Note the user may override this setting during deployment.

From **streisand/playbooks/customize.yml**:

```
6   vars_prompt:
7     - name: streisand_ssh_private_key
8       prompt: "Enter the path to your SSH private key, or press enter for default "
9       default: "~/.ssh/id_rsa"
10      private: no
```

Recommended Remediation:

The assessment team recommends generating a new SSH key which will only be used by Streisand. Do include information in the public key which can be traced back to the user.

References:

A3-Sensitive Data Exposure

L3: [AWS] EC2 Instance Metadata Accessible to VPN and Proxy Users

Description:

The Streisand application permitted proxy users to access EC2 instance metadata. If an IAM role was assigned to an EC2 instance, temporary AWS credentials could have been obtained.

AWS allows EC2 instance to be configured with an associated IAM role. EC2 instances may obtain temporary access keys for their configured IAM role by performing an HTTP request to the URI http://169.254.169.254/latest/meta-data/iam/security-credentials/<ec2IamRole>. Note the temporary access keys are rotated automatically by AWS, which is intended to alleviate developers from key management. Accessing instance metadata does not require authentication, allowing any application installed on the EC2 to obtain metadata about the instance.

Although Streisand did not make use of IAM roles for EC2 instances, advanced users may configure their own EC2 instances with IAM roles. The following proof of concept demonstrates how IAM credentials could have been obtained by proxying traffic through **Shadowsocks**.

Proof of Concept

1. The **Shadowsocks** application was used to connect to the server.

```
./shadowsocks2-linux-x64 -c 54.241.223.91:8530 -password "**REDACTED**" -socks
localhost:1080 -verbose -cipher chacha20-ietf-poly1305
```

1. The **IAM** security role associated with the EC2 instance was obtained through the socks proxy created by **Shadowsocks**. This was achieved by sending a proxied HTTP request to the EC2 instance metadata service http://169.254.169.254/latest/meta-data/iam/info/ using curl. The returned JSON response contained the EC2 IAM role name **iam-aws_administrator_role**.

```
curl --socks5 localhost:1080 http://169.254.169.254/latest/meta-data/iam/info/
{
  "Code" : "Success",
  "LastUpdated" : "2018-03-24T18:11:20Z",
  "InstanceProfileArn" : "arn:aws:iam::414440234058:instance-profile/iam-
aws_administrator_role",
  "InstanceProfileId" : "AIPAJ6GJUHYEMCKVWD34E"
}
```

2. Temporary IAM access keys were obtained by sending a proxied HTTP request to the EC2 metadata service. As shown below, the JSON fields **AccessKeyId**, **SecretAccessKey**, and **Token** contained the access keys. Note the keys have been redacted from the output.

```
curl --socks5 localhost:1080 http://169.254.169.254/latest/meta-data/iam/security-
credentials/iam-aws_administrator_role
{
  "Code" : "Success",
  "LastUpdated" : "2018-03-24T18:12:01Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "**REDACTED**",
  "SecretAccessKey" : "**REDACTED**",
  "Token" : "**REDACTED**",
  "Expiration" : "2018-03-25T00:16:18Z"
}
```

Recommended Remediation:

The assessment team recommends configuring IP routing tables to drop packets to the EC2 metadata instance. This can be achieved with the command **route add -host 169.254.169.254 reject**.

References:

Retrieving Security Credentials from Instance Metadata
Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances

L4: [stunnel] Deprecated TLS Ciphers Supported

Description:

The application's stunnel service uses deprecated algorithms or protocols. Transport Layer Security (TLS) is the encryption component of the stunnel protocol suite. They provide transport-layer security to ensure data confidentiality, authenticity, and integrity. If weak algorithms are not disabled, a suitably located attacker could mount a cipher downgrade attack, forcing both the client and the server to use a cipher weaker than the one they would have normally chosen.

In the context of Streisand, stunnel was only used to tunnel OpenVPN traffic to bypass censorship. In order for a man-in-the-middle attack to occur against OpenVPN traffic passing through stunnel, a vulnerability within the OpenVPN applications or protocol would be required, and would be unrelated to stunnel, ultimately lowering the risk of this issue. Beyond configuration review, additional assessment of OpenVPN and stunnel were out of scope.

Several cipher suites were discovered to not support perfect forward secrecy. Perfect forward secrecy allows previous TLS connections to be secure if the host's private key is compromised. The Ephemeral Diffie-Hellman Key Exchange algorithm is commonly used to achieve perfect forward secrecy.

TLS cipher suites which use 3DES are considered insecure and were used by the application. The cryptographic SWEET32 attack affects ciphers which use a block size of 64 bits, such as 3DES, allowing plaintext to be recovered after 2^32 blocks with the same key have been encrypted. Additionally, cryptographic vulnerabilities of 3DES were found that reduces the cipher's effective key size.

Anonymous cipher suites do not verify the identity of the target host. This allows for a positioned attacker to trivially man-in-the-middle communications.

Protocol	Cipher Suite	Lacks Perfect Forward Secrecy (DHE / ECDHE)	Additional Weaknesses with Cipher Suite or Protocol
TLS 1.1	AECDH-AES256-SHA (TLS_ECDH_anon_WITH_AES_256_CBC_SHA)	X	**X**
TLS 1.1	ADH-AES256-SHA (TLS_DH_anon_WITH_AES_256_CBC_SHA)	X	**X**
TLS 1.1	ADH-CAMELLIA256-SHA (TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA)	X	**X**
TLS 1.1	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)	X	
TLS 1.1	CAMELLIA256-SHA (TLS_RSA_WITH_CAMELLIA_256_CBC_SHA)	X	

TLS 1.1	AECDH-AES128-SHA (TLS_ECDH_anon_WITH_AES_128_CBC_SHA)	X	**X**
TLS 1.1	ADH-AES128-SHA (TLS_DH_anon_WITH_AES_128_CBC_SHA)	X	**X**
TLS 1.1	ADH-CAMELLIA128-SHA (TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA)	X	**X**
TLS 1.1	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)	X	
TLS 1.1	CAMELLIA128-SHA (TLS_RSA_WITH_CAMELLIA_128_CBC_SHA)	X	
TLS 1.2	AECDH-AES256-SHA (TLS_ECDH_anon_WITH_AES_256_CBC_SHA)	X	**X**
TLS 1.2	ADH-AES256-GCM-SHA384 (TLS_DH_anon_WITH_AES_256_GCM_SHA384)	X	**X**
TLS 1.2	ADH-AES256-SHA256 (TLS_DH_anon_WITH_AES_256_CBC_SHA256)	X	**X**
TLS 1.2	ADH-AES256-SHA (TLS_DH_anon_WITH_AES_256_CBC_SHA)	X	**X**
TLS 1.2	ADH-CAMELLIA256-SHA (TLS_DH_anon_WITH_CAMELLIA_256_CBC_SHA)	X	**X**
TLS 1.2	AES256-GCM-SHA384 (TLS_RSA_WITH_AES_256_GCM_SHA384)	X	
TLS 1.2	AES256-SHA256 (TLS_RSA_WITH_AES_256_CBC_SHA256)	X	
TLS 1.2	AES256-SHA (TLS_RSA_WITH_AES_256_CBC_SHA)	X	
TLS 1.2	CAMELLIA256-SHA (TLS_RSA_WITH_CAMELLIA_256_CBC_SHA)	X	
TLS 1.2	AECDH-AES128-SHA (TLS_ECDH_anon_WITH_AES_128_CBC_SHA)	X	**X**
TLS 1.2	ADH-AES128-GCM-SHA256 (TLS_DH_anon_WITH_AES_128_GCM_SHA256)	X	**X**
TLS 1.2	ADH-AES128-SHA256 (TLS_DH_anon_WITH_AES_128_CBC_SHA256)	X	**X**
TLS 1.2	ADH-AES128-SHA (TLS_DH_anon_WITH_AES_128_CBC_SHA)	X	**X**
TLS 1.2	ADH-CAMELLIA128-SHA (TLS_DH_anon_WITH_CAMELLIA_128_CBC_SHA)	X	**X**
TLS 1.2	AES128-GCM-SHA256 (TLS_RSA_WITH_AES_128_GCM_SHA256)	X	
TLS 1.2	AES128-SHA256 (TLS_RSA_WITH_AES_128_CBC_SHA256)	X	
TLS 1.2	AES128-SHA (TLS_RSA_WITH_AES_128_CBC_SHA)	X	
TLS 1.2	CAMELLIA128-SHA (TLS_RSA_WITH_CAMELLIA_128_CBC_SHA)	X	

The below **stunnel** configuration file was used to specify which ciphers to use. Note as the **ciphers** setting is set to **HIGH**, the list of cipher suites to use was delegated to **OpenSSL**, which contains an internal list of cipher suites regarded as **HIGH**. When testing, **OpenSSL 1.0.2g** was found to be installed.

From **streisand/playbooks/roles/stunnel/templates/stunnel-remote.conf.j2**:

```
7 ciphers = HIGH
```

Additionally, the **stunnel** server daemon did not have a cipher preference, allowing connecting clients to determine which ciphers to use.

Recommended Remediation:

The assessment team recommends reconfiguring the stunnel service to ensure that attempts to negotiate anything but the latest TLS version are rejected. Reconfigure cipher suites to ensure only secure configurations are used. The application **testssl.sh** may be used to aid in TLS configuration review.

References:

STunnel Cipher List and Qualys SSL Labs Testing
Testing for SSL-TLS (OWASP-CM-001) – OWASP
Are You Ready for 30 June 2018? Saying Goodbye to SSL/early TLS
NIST Special Publication 800-52 Revision 1 – Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations
Development and Implementation of Secure Web Applications
Sweet32: Birthday attacks on 64-bit block ciphers in TLS and OpenVPN
Here Come the XOR Ninjas
Lucky Thirteen: Breaking the TLS and DTLS Record Protocols

L5: [Monit] Administration Page Accessible Through Proxies

Description:

The Streisand application permitted proxy users to access the **monit** administrative web applications. Users authenticated to the proxy and VPN services could then disable proxy and VPN services.

The **monit** service was configured by Streisand to auto-restart Nginx, Proxies, and VPNs daemons, in case they terminated unexpectedly. The **monit** service administrative HTTP web application was bound to localhost on port **2812**. Users who were proxied through the Streisand host, such as through a Socks proxy created by **OpenSSH**, were able to access the **monit** administrative page. From there, users can disable Proxy and VPN services.

Enabling the **monit** web server was specified in following configuration template:

From **streisand/playbooks/roles/monit/templates/monitrc.j2**:

```
7 set httpd port 2812 and
8     use address localhost  # Bind monit to localhost
9
10     # Allow any ip to connect to monit. Binding monit to localhost
11     # and requiring basic auth (through nginx) should be sufficient to protect this
interface.
12     allow 0.0.0.0/0.0.0.0 # allow any ip to connect to monit.
```

The following proof was used to test the finding.

1. Connect to OpenSSH.

OpenSSH was used to create a SOCKS5 proxy that would tunnel traffic through the Streisand host.

```
ssh -ND 8080 forward@54.241.223.91 -i /tmp/streisand
```

2. Send a POST request to monit to disable sslh

Using the local socks proxy created by **OpenSSH**, a HTTP POST request was sent to the **monit** HTTP server. The request disabled the **sslh** service.

```
curl --socks5 localhost:8080 -D - 127.0.0.1:2812/sslh -d"action=stop"
HTTP/1.0 200 OK
Date: Tue, 27 Mar 2018 19:48:39 GMT
Server: monit 5.16
Content-Length: 4798
Connection: close
Content-Type: text/html
<!DOCTYPE html><html>
...
<td>Status</td><td><span class='gray-text'>Not monitored - stop pending</span></td>
...
```

Recommended Remediation:

The assessment team recommends not enabling the HTTP server built into **monit**. Based on documentation, this can be achieved by not specifying the **set httpd** directive.

References:

OWASP Top 10-2017 A6-Security Misconfiguration
Documentation For MONIT HTTPD Directive

L6: [Gateway] Streisand Gateway Was Fingerprintable

Description:

The Streisand Gateway presents a X.509 certificate with the organization name Streisand. When a user attempts to connect to the gateway, deep packet inspection could be used to view certificate details, and then terminate the connection in order to enforce censorship.

The following configuration shows where the **Organization** and **OrganizationUnit** names were specified in the X.509 certificate provided by nginx.

From **streisand/playbooks/roles/streisand-gateway/defaults/main.yml**:

```
5 nginx_key_org: "Streisand"
6 nginx_key_ou: "Streisand Effect Department"
```

Proof of Concept

A Streisand instance was deployed to 54.241.223.91. The following command was executed to obtain certificate details from Streisand's Gateway.

```
openssl s_client -showcerts -connect 54.241.223.91:443
```

The following was contained in the output, which contained the fingerprintable Streisand text.

```
Server certificate
subject=/C=US/ST=California/L=Malibu/O=Streisand/OU=Streisand Effect Department/CN=Streisand
at 54.241.223.91
issuer=/C=US/ST=California/L=Malibu/O=Streisand/OU=Streisand Effect Department/CN=Streisand
54.241.223.91 Root CA
```

Additionally, an adversary could fingerprint sizes of web pages and VPN clients served by the Streisand Gateway. By monitoring TLS communications and traffic flow, an adversary may be able to determine the user is browsing content on a Streisand gateway.

Due to time constraints, other services were not checked to see if they were easily fingerprintable.

Recommended Remediation:

The assessment team recommends ensuring information which can use to identify the Streisand gateway is not sent in plaintext, such as the Organization property of the certificate being set to **Streisand**. To prevent fingerprinting of the Streisand gateway based on X.509 certificates, properties containing identifying information should contain values which are randomly generated during deployment.

References:

Manual Page for openssl-req

L7: [Gateway,LT2P/IPSEC] Password Generation uses Low Entropy Sources for Randomness

Description:

The Streisand application used the GNU program **shuf** to produce passwords. Per GNU's documentation, the **shuf** program by default uses an internal pseudo-random generator initialized by a small amount of entropy for creating numbers. As low entropy is used for the generator's seed, an attacker may be able to discover the original seed value more easily, allowing for passwords to be determined.

Based on source review, the GNU **shuf** application uses the **ISAAC** cryptographic pseudo random number generator when no **—random-source** argument is specified. The ISAAC generator is partially seeded with a few bytes from **/dev/urandom**, and then may use uninitialized memory, the current time of day, the process **pid**, and a few other weak sources of entropy for additional values for the seed.

The following shows where **shuf** is used to generate passwords and keys for the Streisand gateway (**gateway**), LT2P (**weak_password)**, and IPSec (**psk**).

From **src/streisand/playbooks/roles/common/vars/main.yml**:

```
72   gateway:
73     shuf -n 6 /usr/share/dict/english.txt |
74     paste -s -d '.' -
...
93   weak_password:
94     shuf -n 3 /usr/share/dict/english.txt |
95     paste -s -d '.' -
96   psk:
97     shuf -n 5 /usr/share/dict/english.txt |
98     paste -s -d '.' -
```

Note, the review of **shuf** program was out of scope and thus was not thoroughly reviewed.

Recommended Remediation:

The assessment team recommends ensuring passwords are randomly generated, and seed values used by cryptographic pseudo random number generators are not guessable. To

improve the randomness of **shuf**, use the argument **—random-source** with a value of **/dev/urandom**.

References:

GNU Coreutils: Random sources
ISAAC seed source code used by shuf
SEI Cert C Coding Standard- MSC32-C. Properly seed pseudorandom number generators

L8: Tor Browser Bundle Uses Default Security Settings

Description:

The Streisand gateway provides a mirror of the Tor Browser Bundle version 7.5.2. When downloaded and installed on Linux, multimedia and JavaScript support were enabled. Processing of multimedia files or JavaScript may lead to a user's system becoming compromised.

When hardening a system, it's best to disable features which will not be used. Web browsers notably have a large attack surface, as many features are enabled by default. Exploiting memory corruption vulnerabilities in the processing of multimedia files and JavaScript have been used in the past to gain remote code execution.

For instance, on March 13th, 2018, Firefox issued a security advisory stating Firefox ESR 52.7 contained a fix for a buffer overflow which could be triggered while processing a malicious SVG file. A new version of the Tor Browser Bundle was then released to address the issue, as the Tor Browser is based off Firefox.

Note that Streisand did not explicitly enable multimedia and JavaScript support. These are the default settings of the Tor Browser Bundle. The Tor Project's site states: “We configure NoScript to allow JavaScript by default in Tor Browser because many websites will not work with JavaScript disabled.”

Proof of Concept

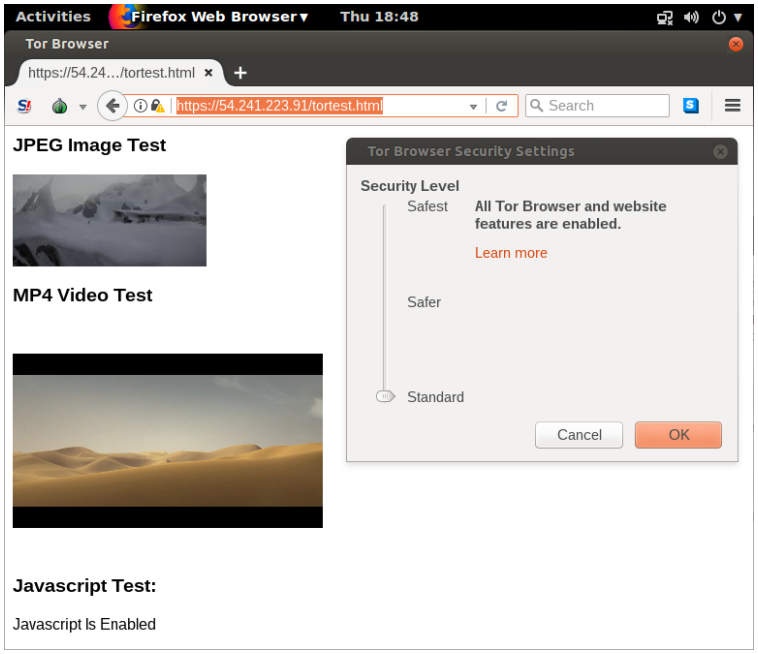
The following proof of concept was used to test multimedia and JavaScript support in the default configuration of the Tor Browser Bundle.

1. The following was copied to the Streisand gateway file path **/var/www/streisand/toretest.html**.

```
<html>
<body>
<h3>JPEG Image Test</h3>
<img src="https://durian.blender.org/wp-content/uploads/2010/06/01a_comp_000559-200x95.jpg" />
<h3>OGG Video Test</h3>
<video width="320" height="240" controls> <source
src="https://download.blender.org/durian/trailer/sintel_trailer-480p.ogv"> </video>
<h3>JavaScript Test: </h3>
<p id="jstest">JavaScript Is Disabled</p>
<script>document.getElementById("jstest").textContent = "JavaScript Is Enabled";</script>
</body>
</html>
```

2. The Tor Browser Bundle was obtained through Streisand's gateway through the URL https://54.241.223.91/mirror/tor/tor-browser-linux64-7.5.2_en-US.tar.xz.
3. The Tor Browser Bundle instructions were followed per the URL https://54.241.223.91/tor/#desktops.
4. Using the Tor Browser, the assessment team navigated to the URL https://54.241.223.91/tortest.html. The rendered page displayed the JPEG and OGG video which was linked to in the document. Additionally, the JavaScript embedded in the page executed, which updated the page to display **JavaScript Is Enabled**. Additionally, Tor's default security settings were observed to be set to **Standard**. This was determined by clicking the Tor icon, which is next to the browser's address bar, followed by clicking **Security Settings**.

The following is a screenshot of the Tor Browser Bundle rendering the page, as well as the Tor Browser Bundle's default security settings.



Recommended Remediation:

The assessment team recommends providing a hardened Tor Browser Bundle configuration. Disable JavaScript, multimedia support, and custom fonts if possible. As Tor exit nodes may inspect traffic, ensure HTTPS is used whenever possible.

References:

Why is NoScript configured to allow JavaScript by default in Tor Browser? Isn't that unsafe?
Mozilla Foundation Security Advisory 2018-07
[tor-announce] Tor Browser 7.5.1 is released

Artwork obtained from the Durian Open Movie project Sintel.

APPENDICES

A1: Credential Complexity, Usage, and Storage Considerations

Some potential issues and hardening recommendations related to key and password generation, storage, and usage are described in further detail below:

Cryptographic Keys and Passwords Stored as Plain Text

Documentation generated by Streisand contains passwords for services which are stored as plain text on the server. Additionally, OpenVPN certificates are not password-protected. If an attacker compromises the Streisand server or a client's computer, credentials can be easily obtained. Due to time constraints, the full impact of this issue was not investigated. The assessment team recommends encrypting sensitive data whenever possible.

Cryptographic Keys and Password Viewable Between Users

The Streisand gateway uses a single set of Basic Authentication credentials to authenticate users. After authenticating with the gateway, credentials for any user profile can be obtained. As a best practice, each user should obtain their own credentials, and should not be able to obtain passwords, keys, or configuration parameters used by other clients. Due to time constraints, the impact of this behavior was not investigated.

Review Credential Complexity

Services provided by Streisand may rely on a pre-shared key. Keys which are not sufficiently complex may be susceptible to brute-force attacks. Generally speaking, encryption keys and password hashes should not realistically be brute-forceable by attackers. Brute forcing credentials over a network should be at least as costly as local brute force attempts. Due to time limitations and scope, the complexity involved in brute forcing of credentials and encryption keys were not thoroughly investigated.

One instance where a weak shared key is in use is LT2P CHAP. Note that the CHAP protocol is vulnerable to offline brute-force attacks for its secret shared key. As a single round of SHA1 or MD5 is used by the protocol to authenticate requests, a shared secret containing 128 bits of entropy is recommended. Based on source review, the password used by CHAP was equivalent to 33 bits of security, or **log2(2048^3)** where 2048 was the number of English words in **english.txt** and 3 was the number of random words to select.

The following shows the generation of the CHAP shared key. From **src/streisand/playbooks/roles/common/vars/main.yml**:

```
93   weak_password:
94     shuf -n 3 /usr/share/dict/english.txt |
95     paste -s -d '.' -
```

Rotate Credentials and Encryption Keys

Rotate encryption keys and credentials periodically. Performing this action helps to mitigate the risk of an attacker authenticating with Streisand or eavesdropping on communications. Due to scoping, rotation of passwords and keys was not investigated.

Review Authentication Limits

Attackers often use brute-force attempts to discover valid credentials to a system. As a best practice, authentication attempts should be monitored. Restrict access to services upon multiple unsuccessful authentication attempts and if possible, notify the owner of the account. Due to the time-boxed nature of the assessment, rate limiting of authentication requests was not investigated.