# pfSense **Community Edition Firewall** Final Report

|  |  |
|---|---|
| ***Document Name:*** | Final Report |
| ***Date:*** | July 13, 2017 |
| ***Customer Contact:*** | Joe Basirico <jbasirico+research@securityinnovation.com> |
| ***Author:*** | Matthew Halchyshak <mhalchyshak@securityinnovation.com><br>Tyson Nieuwenhuis <tnieuwenhuis@securityinnovation.com><br>Anthony Follett <afollett@securityinnovation.com><br>Izzy Gambliel <egambliel@securityinnovation.com><br>Shreeraj Shah <sshah@securityinnovation.com><br>Uday Bhaskar <ubhaskar@securityinnovation.com> |
| ***Project Manager:*** | Garrett Jaynes <gjaynes@securityinnovation.com> |

## Contact Information

### Security Innovation

#### *Business Contact*

Marcus Hodges
Research Director
mhodges@securityinnovation.com

#### *Technical Contact*

Joe Basirico
Senior VP of Engineering
jbasirico@securityinnovation.com
Mobile: +1.206.227.6458

#### *Project Management Contact*

Garrett Jaynes
Senior Project Manager
gjaynes@securityinnovation.com

### pfSense

Joe Basirico
VP of Services
jbasirico+research@securityinnovation.com

## Executive Summary

Security Innovation performed a thorough security audit of pfSense Community Edition Firewall Version 2.3.3 built on FreeBSD 10.3. This report summarizes the testing that was performed and the issues that were uncovered.

The pfSense Community Edition Firewall is an open source firewall/router based on FreeBSD. The Community Edition Firewall is configurable from the operating system console or through a web based management interface. It offers a range of features such as firewall and routing functionality (NAT, packet scrubbing, Layer 2 bridging) and connectivity (VPN, QoS, Radius). Support for various packages such as Snort, Squid, and haproxy, is also included. pfSense Community Edition Firewall is typically deployed as a perimeter firewall or router with typical network functionality such as DNS, DHCP, etc.

Testing of the application was focused on the core functionality of the pfSense Community Edition Firewall web interface, although some testing was also performed against the XMLRPC and command-line interfaces. Security Innovation did not examine the underlying operating system, plugins, or additional services run by the system. Security Innovation also looked to ensure that the system maintains secure default options while testing the default deployment of Community Edition Firewall .

There were few serious issues that were identified in pfSense Community Edition Firewall that could enable an attacker to steal user sessions through the pfSense Community Edition Firewall Web Interface. Also Security Innovation found a lack of password management policies for Web and Command Line interfaces.

Major observations are as follows:

- A total of 4 security issues were identified:
  - Problem Report 1 - Lack of Password Policy
  - Problem Report 2 - Missing Brute Force Protection
  - Problem Report 3 - Cross-Site Request Forgery
  - Problem Report 4 - Cross-Site Scripting (XSS)
- The most common type of security vulnerability found was lack of brute force deterrence, which can be exploited through user enumeration and password brute force attacks.
- Multiple severe vulnerabilities such as Cross-Site Scripting, Cross-Site Request Forgery were identified.
- The issues fall into all STRIDE (**Spoofing, Tampering, Repudiation, Information Disclosure, Denial Of Service and Elevation Of Privilege**) categories.
- If left unfixed, the reported issues can allow an attacker to gain complete access to a pfSense instance.

An overview of the above issues and their impact is provided in the Problem Summary section of this report, and the details of each issue are provided in self-contained problem reports located in the Problem Reports section.

## Introduction

Security Innovation created a threat model, wrote a comprehensive test plan, and performed security testing of pfSense Community Edition Firewall .

Exploratory tests were performed to gain an understanding of the system that cannot be obtained through public knowledge or specification documents. Next, a threat model was created to explore assets, threats, attack vectors and conditions required for successful attack. Finally a test plan was developed to guide the attack and test execution process, ensuring every avenue of attack was thoroughly covered.

**The top potential risks discovered during system threat modeling were:**

- An Anonymous or Unprivileged user can create, update or delete Authentication Credentials.
- An Anonymous or Unprivileged user can create, read, update or delete Network Traffic.
- An Anonymous or Unprivileged user can create, read, update or delete Configuration Data.
- An Anonymous or Unprivileged user can create, read, update or delete Logs.

**The attacks in the test plan and the subsequent testing were focused on the following areas:**

- Authentication and access control checks throughout the application
- Injection attacks that could be used to extract data or perform Remote Code Execution

This report provides a summary of the observations made while Penetration testing of Community Edition Firewall . The "Threat Analysis" section goes into more depth on the initial analysis work performed. The "Problem Report Summary" section summarizes the issues discovered during the engagement. The "Tools" section describes the tools used in testing. The "Executed Test Cases" section describes all testing that was performed on the component together with the observed results. The "Future Testing" section provides our recommendations on additional future testing on this system. The "Problem Reports" section contains the full text of each bug report.

## Threat Analysis

This section describes the results of our threat analysis of Community Edition Firewall . This threat analysis was one of the factors used to guide the code review process.

**While investigating, the following assumptions were made:**

- The pfSense web application was installed with default settings and configurations.
- The source code of the application was used to aid testing; however, a full-fledged source code review was not performed.
- Issues related to frameworks, plugins, and external services issues were out of scope

**The following characteristics impact attack surface:**

- The application needs root access to perform many of its tasks.
- The application interfaces are only accessible from an internal network or with local access.
- The application contains file upload functionality.
- The application has multiple interfaces includes web and command line applications.
- The web interface has a very granular permission system that allows for granting access to specific pages.

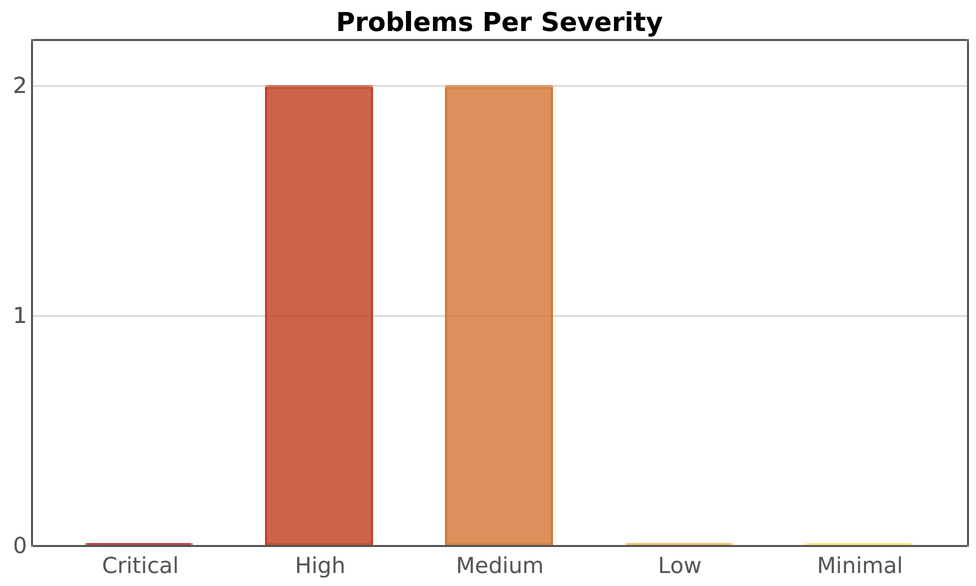
**The following assets should be protected:**

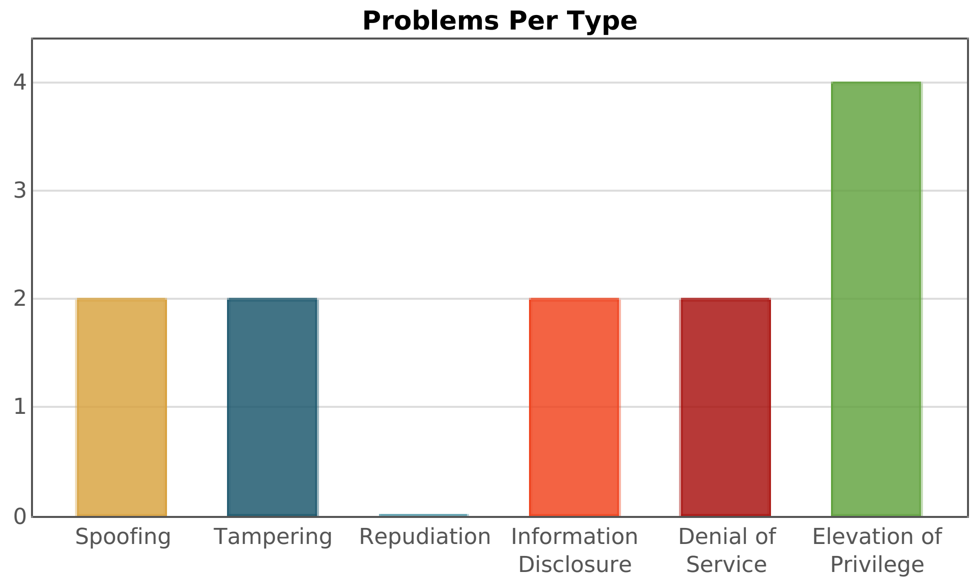
- Authentication Credentials - Credentials used for authentication and role validation t
- Session Tokens - Tokens used for authorization with in the web application
- Configuration Data - Data that describes the current configuration of the firewall
- User Information - Any user information contained within the web application
- Certificates - Certificates uploaded to the server for TLS or authentication
- SSH Keys - SSH keys uploaded to the server
- System Files - Any files within the file system of the server
- Network Traffic - Any network traffic that is processed by or logged within the server
- Logs - Any log files generated by components of the system

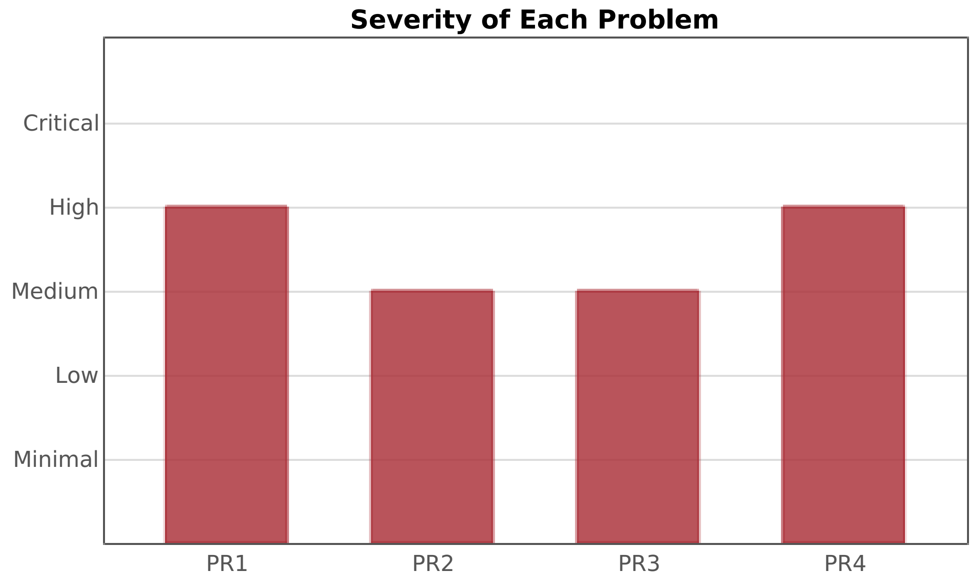
Please see the document titled "pfSense Threat Model" for more details and an exhaustive list of threats against the system.

## Problem Report Summary

A total of 4 problems were identified. This section describes, at a high level, each of the problems discovered. See the Problem Summaries section for a table of each problem discovered, its severity, description and consequences. The following charts display the number of problems for each level of severity, the number of problems for each STRIDE type (note that a problem may have more than one type), and each problem's overall severity.







## Problem Summaries

The problem report summaries are sorted by problem report ID. The format of the problem report table is as follows:

- The problem report ID
- The component in which the issue was discovered
- The severity of the issue
- A short description of the issue
- The consequences of the issue

| PR # | Component | Severity | Description | Consequence |
|---|---|---|---|---|
| 1 | pfSense Web Interface | High | pfSense Community Edition allows users to set weak passwords when creating an account or resetting an account password. | An attacker can more easily obtain access to a targeted account via brute forcing a weak password or by guessing the correct value. |
| 2 | pfSense Web Application | Medium | Community Edition Firewall does not implement measures to limit authentication requests after multiple failed attempts. | The application is more susceptible to brute force and password guessing attacks. An attacker that successfully guesses a user's password will gain full access to that user's account. |

| PR # | Component | Severity | Description | Consequence |
|---|---|---|---|---|
| 3 | pfSense Web Interface | Medium | The application is vulnerable to Cross-Site Request Forgery attacks. | An attacker can coerce a legitimate, authenticated user into performing allowed actions on behalf of the attacker without the user's consent or knowledge. |
| 4 | pfSense Web Interface | High | The application does not properly validate input or output encode user provided input. | An attacker can steal a user's cookies and impersonate the user, coerce the user into carrying out unintended actions on the web application, or modify the content displayed to the user. |

## Tools

In testing the pfSense Community Edition Firewall the following tools were employed:

| Tool | Description | Link |
|---|---|---|
| **Burp Suite Professional** | Interactive HTTP/S proxy server that can intercept, inspect and modify data between the browser and target web server | https://portswigger.net/burp/ |
| **Wireshark** | Gold standard of network sniffers and analysis | https://www.wireshark.org/ |
| **Curl** | Command line tool for transferring files with URL syntax, supporting FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS and FILE | https://curl.haxx.se/ |
| **Python** | High level, general purpose, scripting language | https://www.python.org |
| **Testssl.sh** | testssl.sh is a Command line tool for checks a server's service on any port or the support of TLS ciphers, protocols and recent cryptographic flaw. | https://testssl.sh/ |

## Executed Test Cases

The following table shows the breakdown of executed test cases, including any problem reports relevant to that item, and gives a brief summary of the methodology used to check that item and any other observations.

Column descriptions are as follows:

- ID - An identifier for quick test case reference
- Title - A title describing the test case
- Description - A short description of the test case and why it was performed
- Outcome - Either 'Pass' or a reference to the Problem Report Number

| ID | Title | Description | Outcome |
|---|---|---|---|
| 1 | TLS Configuration | Verify that the TLS configuration does not allow for insecure protocols, cipher suites, or features. | **Pass** |
| 2 | Transport Security of All Traffic | Confirm that all traffic, regardless of the sensitivity of the data, is protected by TLS. | **Pass** |
| 3 | Verbose HTTP Headers | Verify that version information or other sensitive information is not disclosed through headers. | **Pass** |
| 4 | CSRF Protection | Confirm that CSRF tokens are used while invoking any methods that change server-side data or session state. | **Failed (PR 3)** |
| 5 | Verbose Error Messages | Confirm that no invalid input triggers errors with information useful for attacking the system, such as stack traces, or path information. | **Pass** |
| 6 | Direct Request | Confirm that resources are protected by authorization controls and not accessible just by the resource identifier. | **Pass** |
| 7 | Session Token in GET | Confirm that session tokens are not used as GET parameters. | **Pass** |
| 8 | Session Token GET parameter trust | Confirm that the application does not trust session tokens provided as GET Parameters. | **Pass** |
| 9 | Session Token Fixation | Confirm that successfully logging in creates a new session token, preventing session fixation. | **Pass** |
| 10 | Session Expiration Invalidation | Confirm that the system has time or inactivity based session logout and that expired session tokens cannot be reused. | **Pass** |
| 11 | Manual Session Invalidation | Confirm that manually logged out session tokens cannot be reused. | **Pass** |
| 12 | Plaintext Transmission of Credentials | Confirm that all login pages are protected by TLS. | **Pass** |
| 13 | Session Token Predictability | Confirm that session tokens are generated with sufficient entropy as to be unpredictable. | **Pass** |

| ID | Title | Description | Outcome |
|---|---|---|---|
| 14 | Session Token Timing Attacks | Confirm that the validation of session tokens is not vulnerable to timing attacks. | **Pass** |
| 15 | User Enumeration | Confirm that queries that use a username or email which are in the data store return identical responses regardless of whether usernames or emails are in the database. | **Pass** |
| 16 | HTTP Methods | Confirm that insecure HTTP methods such as TRACE or TRACK are disabled and that other methods are either disabled or apply the same authorization checks as GET and POST. | **Pass** |
| 17 | Known Vulnerabilities | Confirm that any software with disclosed version information is not vulnerable to known security vulnerabilities. | **Pass** |
| 18 | Password Storage | Confirm that no functionality indicates the insecure storage of passwords, such as overly restrictive password character or length limitations, or the ability to recover existing passwords. | **Pass** |
| 19 | Reflected Cross-Site Scripting | Confirm that no user input is reflected back in a way that allows for modification of the DOM or script execution. | **Failed (PR 4)** |
| 20 | Stored Cross-Site Scripting | Confirm that no user input is stored and reflected back in a way that allows for modification of the DOM or script execution. | **Failed (PR 4)** |
| 21 | SQL Injection | Confirm that queries sent with SQL meta characters do not cause recognizable errors via error messages, codes, or reproducible timing differences. | **Pass** |
| 22 | Log File Injection | Verify that special characters such as line feeds, carriage returns and scripting characters like < & > cannot be, or are encoded before being, written to log files. | **Pass** |
| 23 | Race Conditions | Confirm that the application does not have any concurrency problems that may be used by an attacker to yield unexpected results. | **Pass** |
| 24 | Mass Assignment | This test focuses on discovering if the application is performing the assignment of multiple values to attributes via a single assignment operator in a safe way. | **Pass** |
| 25 | Sensitive Data over URLs | Confirm that sensitive data is not passed via URL query string parameters (i.e. should be done via POST statement over TLS). Also verify if an attacker can edit these parameters and the application accepts these changes. | **Pass** |

| ID | Title | Description | Outcome |
|---|---|---|---|
| 26 | Logging | Confirm that no confidential information is being written to logs (e.g. application logs, plists, mobile device logs including any debugging that is enabled). | **Pass** |
| 27 | Cryptography strength | Verify that the current cryptography in place matches the industry best standards. | **Pass** |
| 28 | Password Strength Policy | Confirm that there is a configurable method to control password strength. | **Failed (PR 1)** |
| 29 | Cookie Security | Confirm that all cookies are using all appropriate flags, paths, and expirations. | **Pass** |
| 30 | Open Redirects | Confirm that user input is not insecurely used in HTTP or JavaScript redirects. | **Pass** |
| 31 | Path Traversal | Confirm that internal directories and their contents cannot be accessed by manipulating the URL or any parameters. | **Pass** |
| 32 | Authentication Vulnerable to Brute Forcing | Confirm that there is a mechanism in place to hinder brute force attempts such as a Captcha or login throttling. | **Failed (PR 2)** |
| 33 | Client-Side Validation | Confirm that all client-side validation is performed as strictly or more strictly on the server. | **Failed (PR 4)** |
| 34 | Vulnerable File Upload | Confirm that if the application contains file upload functionality, that an attacker cannot upload a file with malicious content to the server. | **Pass** |
| 35 | Buffer Overflow | Ensure proper checks are in place before allowing write on to the buffer. Verify if the application is using insecure functions which are vulnerable to Buffer overflow attacks. | **Pass** |
| 36 | Insecure random number generator | Confirm if arc4random or any weak random number generator is being used to store sensitive data. | **Pass** |
| 37 | Insecure Memory error checking | Confirm if the application makes use of malloc insecurely | **Pass** |
| 38 | XML Parser Features | Confirm that unnecessary and insecure features of XML parsing libraries, such as fetching external files, are disabled or properly protected. | **Pass** |
| 39 | Object Injection | Confirm that untrusted input is not unserialized allowed for the creation of attacker controlled objects. | **Pass** |
| 40 | Code Injection | Confirm that malicious user input cannot be injected and the source code retrieved or attacker code executed. | **Pass** |
| 41 | Command Injection | Confirm that user input is not made a part of a system command and run against the OS. | **Pass** |

| ID | Title | Description | Outcome |
|---|---|---|---|
| 42 | Third-Party Information Disclosure | Confirm that no sensitive information is leaked to third-party sites. | **Pass** |
| 43 | Web/App Server Misconfigured | Confirm that the Web Server/App Server is configured securely. | **Pass** |
| 44 | XML Injection | Confirm that untrusted input is not interpolated into XML in such a way as to change the context of data. | **Pass** |
| 45 | Remote File Include | Confirm that any remote file includes based on user input are properly validated. | **Pass** |
| 46 | Session Token in Source | Confirm that session tokens are not written to the page source anywhere. | **Pass** |
| 47 | Clickjacking Protections | Confirm that the application uses X-Frame-Options and other clickjacking mitigations. | **Pass** |
| 48 | Business Logic Attacks | Confirm that the application's business logic functionality cannot be used in unintended ways for malicious purposes. | **Pass** |
| 49 | Access Control Mechanisms | Confirm that all sensitive data and functionality has proper access control mechanisms to restrict usage to authorized users. | **Pass** |
| 50 | Simultaneous Login | Confirm that the application provides a secure mechanism for handling concurrent logins. | **Pass** |
| 51 | HTTP Header Security | Ensure that the appropriate HTTP headers are set. | **Pass** |

## Recommended Next Steps

This section contains our recommendations for areas that may benefit from additional testing. For each section we will describe why it is important to test these sections again, more thoroughly or for the first time.

**Retest after remediation** - A retest of the pfSense Community Edition Firewall system is recommended to be performed when the problems found as a result of this test have been remediated. This validates the remediation put in place, and ensures other vulnerabilities have not been introduced as a part of the remediation.

**Code Review** - During this assessment Security Innovation did have access to source code and used it to assist in our testing. We do, however, recommend a full code review be performed in order to detect more subtle issues that may not be easily discovered through dynamic testing.

## Problem Reports

Below are the complete Problem Reports for all discovered issues.

### Problem Report 1 - Lack of Password Policy

pfSense Community Edition Firewall allows the user to set weak passwords such as a single numeric character or the same password as the username. This allows users to set a weak password, which can more easily be guessed with a brute force or dictionary attack.

| | |
|---|---|
| **Component** | pfSense Web Interface |
| **STRIDE** | Elevation of Privilege |
| **CWE** | CWE-521: Weak Password Requirements |
| **CAPEC** | CAPEC-16: Dictionary-based Password Attack<br>CAPEC-49: Password Brute Forcing |
| **CVSS v2 Score** | 5.8 (AV:A/AC:L/Au:N/C:P/I:P/A:P) |
| **OWASP Reference** | OWASP Top 10 2013-A5: Security Misconfiguration |

| **Overall Severity** | **High** |
|---|---|
| **Vulnerability Type** | **Defense in Depth** |
| **Impact** | **High** |
| **Confidentiality** | An attacker that guesses or derives a user's password would have full access to the account. |
| **Integrity** | There is no direct impact on the data integrity. |
| **Availability** | An attacker that guesses or dervies a password for an administrative account can delete all existing accounts and disable firewall rules. |
| **Exposure** | This enables an attacker to more easily enumerate weak passwords by launching brute force attacks against the users of the application or successfully guessing weak chosen passwords. |
| **Affected Users** | All users of the application are affected. |
| **Likelihood** | **Medium** |
| **Skill Required** | Minimum technical skills are required to launch an attack. However, an automated attack would require scripting skills or knowledge of other password brute forcing tools. |
| **Conditions and Complexity** | Although not necessary, a list of valid usernames would increase the likelihood of a successfuly brute force attack. |
| **Discoverability** | This can be easily discovered by users with access to the application. |
| **Reproducibility** | This is always reproducible. |

#### *Background Information*

The password policies of web applications aim to control the length, complexity, and age of user passwords to deter dictionary and brute force attacks. When a user is allowed to set an insecure password, such as '*abc123*', the account is put in a higher risk of compromise.

Typical attacks against user passwords include dictionary attacks that aim to obtain user passwords by repeatedly trying plausible character combinations, such as educated password guesses based on relevant facts of the target user, common passwords, words, and phrases. Brute force attacks attempt to obtain user passwords by repeatedly trying all possible character combinations until they are exhausted.

### Problem Details

The pfSense Community Edition Firewall does not currently enforce a password policy during account creation or password resets. In particular, the following best practices identified by Security Innovation are not followed:

- There is no minimum length for a password, including single character passwords.
- There is no complexity requirements regarding special or case sensitive characters.
- There is no password history preventing previous passwords from being used.
- There is no rule preventing a username from being used as the password.
- There is no requirement that passwords be changed after the first login.
- There is no dictionary used to prevent the use of common weak passwords or phrases.

#### Affected Areas

The following areas of the application was found to be vulnerable.

**User Management Page**

- https://<pfSense IP Address>/system_usermanager.php?act=edit&userid=0

### Test Steps

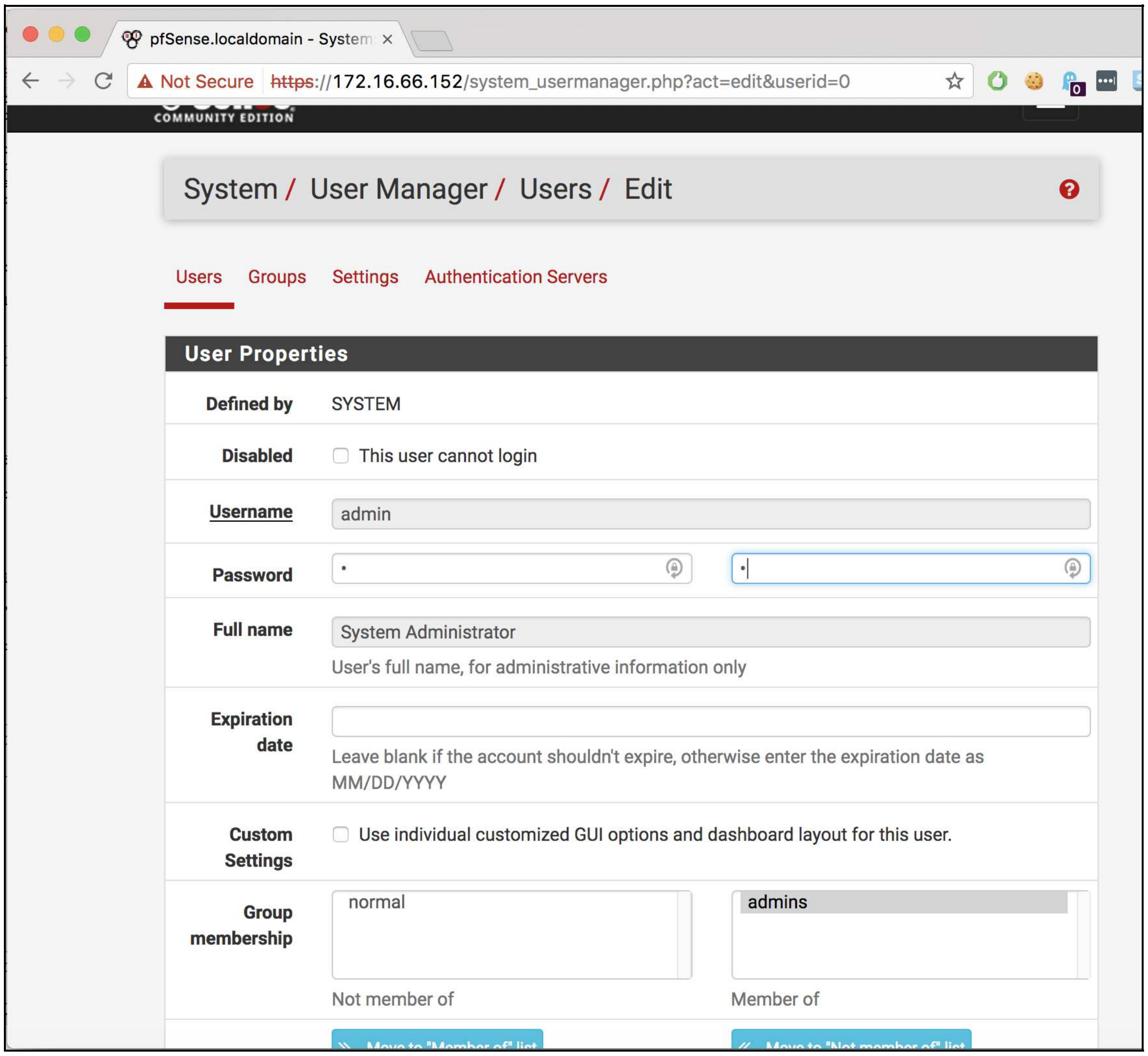
#### Test Configuration

The following is needed in order to reproduce this issue:

- Access to the pfSense application and an Administrative account.

#### Steps to Reproduce

1. Login to the pfSense application with the Administrative account and navigate to `System` -> `User Manager`.
2. Click the `Edit User` button in the `Action` Section.
3. Enter details as shown in the screenshot below. Set a weak password such as `123`, click the `Save` button, and



pfSense.localdomain - System

Not Secure https://172.16.66.152/system_usermanager.php?act=edit&userid=0

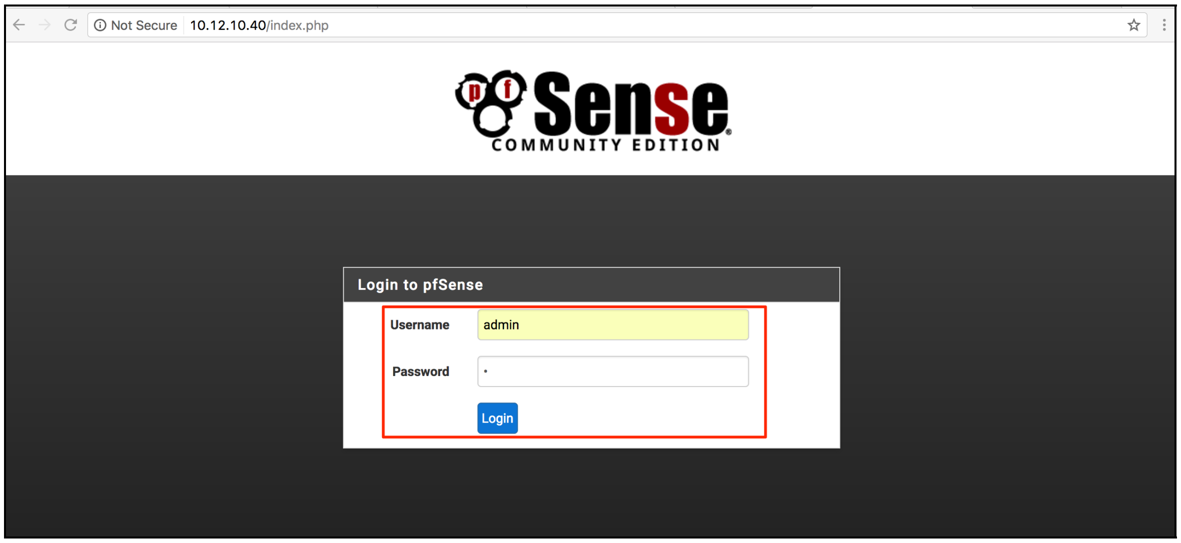
System / User Manager / Users / Edit

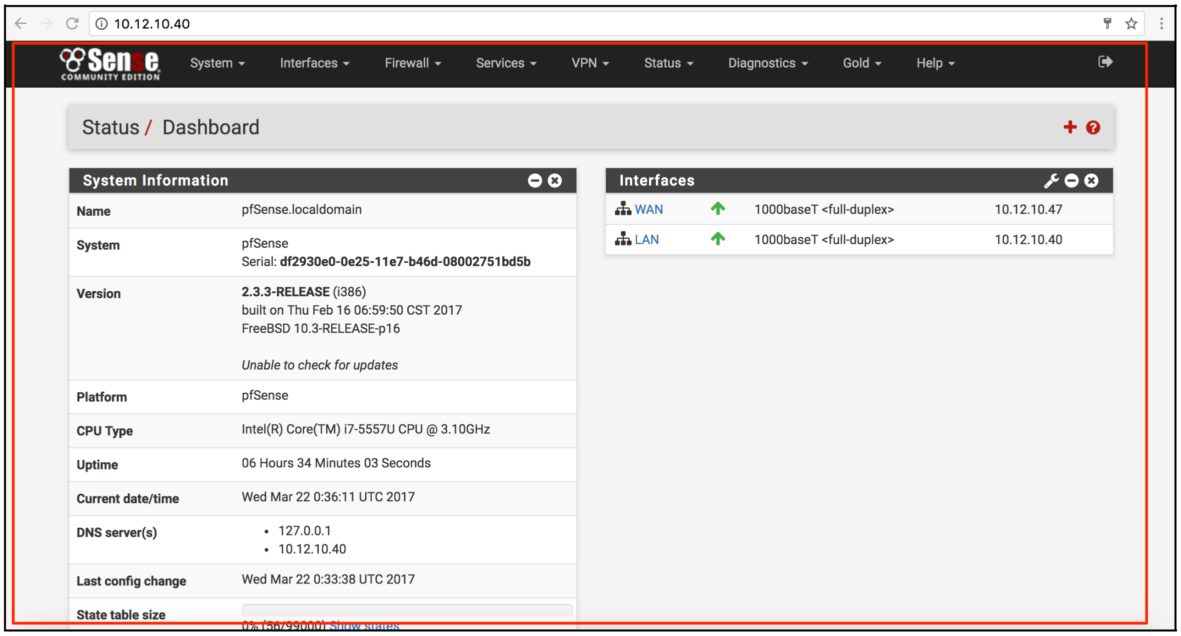
Users Groups Settings Authentication Servers

**User Properties**

| | |
|---|---|
| Defined by | SYSTEM |
| Disabled | ☐ This user cannot login |
| Username | admin |
| Password | • / • |
| Full name | System Administrator<br>User's full name, for administrative information only |
| Expiration date | Leave blank if the account shouldn't expire, otherwise enter the expiration date as MM/DD/YYYY |
| Custom Settings | ☐ Use individual customized GUI options and dashboard layout for this user. |
| Group membership | Not member of: normal<br>Member of: admins |

4. Observe that the application accepts the weak password .




10.12.10.40
System Interfaces Firewall Services VPN Status Diagnostics Gold Help
Status / Dashboard

System Information

| | |
|---|---|
| Name | pfSense.localdomain |
| System | pfSense<br>Serial: **df2930e0-0e25-11e7-b46d-08002751bd5b** |
| Version | **2.3.3-RELEASE** (i386)<br>built on Thu Feb 16 06:59:50 CST 2017<br>FreeBSD 10.3-RELEASE-p16<br>*Unable to check for updates* |
| Platform | pfSense |
| CPU Type | Intel(R) Core(TM) i7-5557U CPU @ 3.10GHz |
| Uptime | 06 Hours 34 Minutes 03 Seconds |
| Current date/time | Wed Mar 22 0:36:11 UTC 2017 |
| DNS server(s) | • 127.0.0.1<br>• 10.12.10.40 |
| Last config change | Wed Mar 22 0:33:38 UTC 2017 |
| State table size | 0% (56/99000) Show states |

Interfaces

| | | | |
|---|---|---|---|
| WAN | ↑ | 1000baseT <full-duplex> | 10.12.10.47 |
| LAN | ↑ | 1000baseT <full-duplex> | 10.12.10.40 |

## *Remediation*

Enforce a strong password policy. At a minimum, passwords should satisfy the following requirements:

- Passwords must contain 8 characters or above, and at least 5 of the characters must be unique.
- Passwords must not contain the user's username, email address or their first or last name.
- Passwords must contain at least 1 character from each character types: lowercase letters, uppercase letters, numbers, and special characters.
- Users must be forced to change their password upon first login.
- Maintain a password history and restrict users from re-using their previous 5 passwords.

- Password policy must be configurable by admin

For the administrative account as a security good practice, consider requiring passwords to contain at least 20 characters and setting a password age. When the password age expires, the user must create a new password.

**References:**

http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf

https://www.owasp.org/index.php/Authentication_Cheat_Sheet

## Problem Report 2 - Missing Brute Force Protection

pfSense Community Edition Firewall does not implement measures to limit authentication requests after multiple failed attempts. This makes the application more susceptible to brute force and password guessing attacks. An attacker who has successfully guessed a user's password will gain full access to that user's account.

| **Component** | pfSense Web Application |
|---|---|
| **STRIDE** | Spoofing, Information Disclosure, Elevation of Privilege |
| **CWE** | CWE-307: Improper Restriction of Excessive Authentication Attempts |
| **CAPEC** | CAPEC-49: Password Brute Forcing |
| **CVSS v2 Score** | 5.4 (AV:A/AC:M/Au:N/C:P/I:P/A:P) |
| **OWASP Reference** | OWASP Top 10 2013-A2 : Broken Authentication and Session Management |

| **Overall Severity** | **Medium** |
|---|---|
| **Vulnerability Type** | **Defense in Depth** |
| **Impact** | **Medium** |
| **Confidentiality** | An attacker with a valid list of usernames can try and guess the password to gain access to a specific user's account. |
| **Integrity** | The compromised user's data could be modified by an attacker. |
| **Availability** | The application could be unavailable to the compromised user. |
| **Exposure** | The application would be exposed to brute force attacks, which could lead to compromise of user accounts. |
| **Affected Users** | All users of the application are affected. |
| **Likelihood** | **Medium** |
| **Skill Required** | No special skills needed but scripting skills or knowledge of brute forcing tools would increase the likelihood of success. |
| **Conditions and Complexity** | A valid list of usernames would need to be enumerated before mounting an automated password guessing attack. Absence of a strict password policy makes this attack easier. |
| **Discoverability** | This can be easily discovered by anyone present on the local network. |
| **Reproducibility** | This can always be reproduced. |

### *Problem Details*

pfSense Community Edition Firewall does not mitigate against brute force and password guessing attacks on the login page. This allows a malicious attacker unlimited attempts at authenticating to a valid user's account without prior knowledge of their password. This increases the likelihood that an attacker can gain access to their account. This vulnerability coupled with other vulnerabilities such as "Lack of password policy" would increase the risk of accounts being compromised.

<u>Affected Areas</u>

**Login Page**

- https://<pfSense IP Address>

### *Test Steps*

<u>Test Configuration</u>

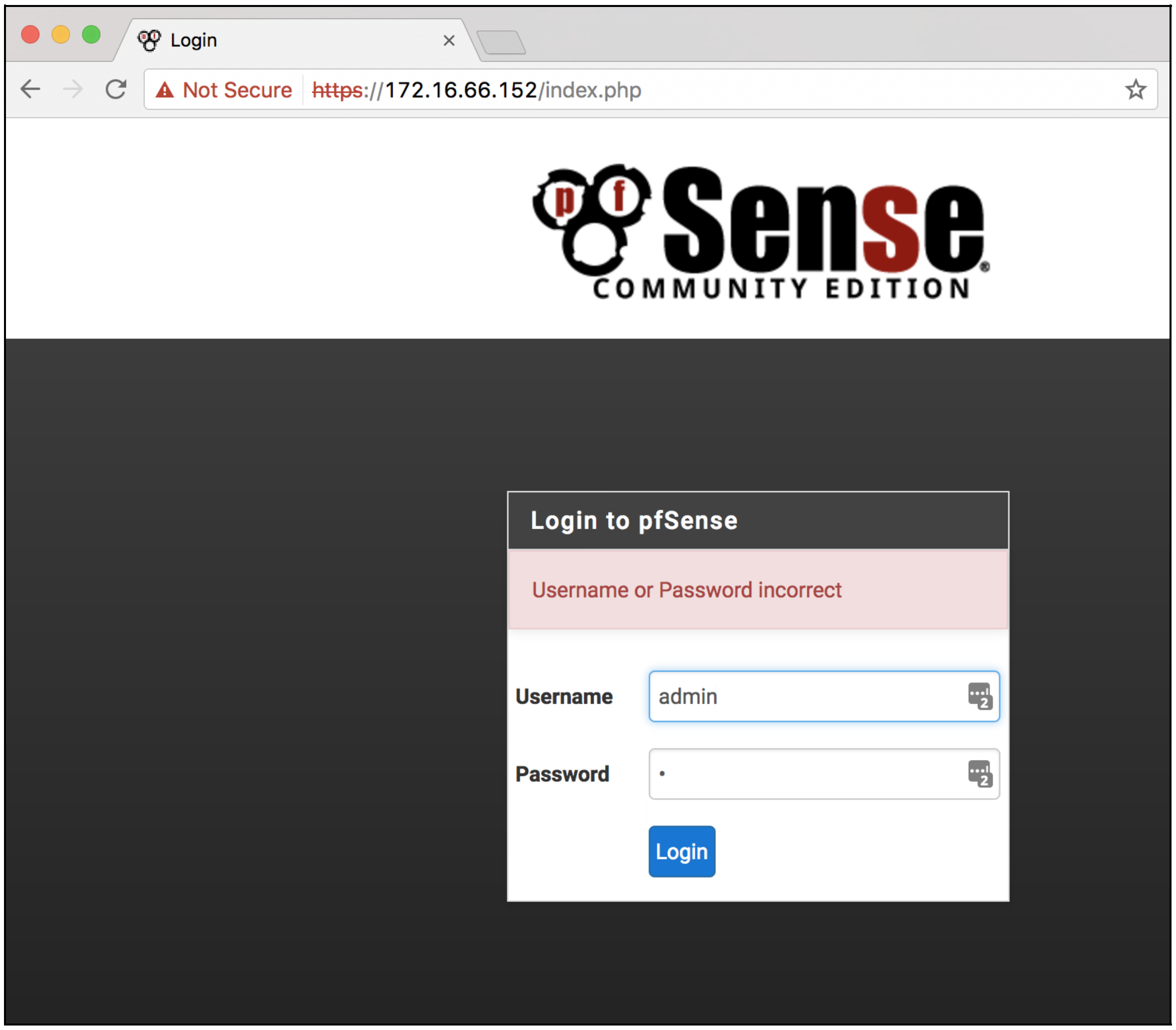
The following is needed in order to reproduce this issue:

- Access to the pfSense Web interface

<u>Steps to Reproduce</u>

1. Access the Login page. Attempt to login using a valid username and invalid password combination. https://<pfSense IP Address>

2. Repeat the above step **15** times. Observe that each login attempt fails with the same error message.


Login
Not Secure https://172.16.66.152/index.php
pfSense COMMUNITY EDITION
Login to pfSense

Username or Password incorrect

Username: admin

Password: •

3. Login with the same username using the correct password. Observe that the user is successfully authenticated to the application.

## *Remediation*

After a sufficient number of unsuccessful login attempts have occurred, one of the following actions, in order of preference below, should be taken.

- **Progressively Throttle Authentication Attempts** - Ensure that after a small number of failed attempts, each consecutive attempt take more time than the previous attempt to complete. The increased delay introduced after each failed attempt will prevent automated tools from attempting a large number of guesses, prevents an attacker from locking out valid users due to failed login attempts, and provides usability for users to recover easily from a small number of legitimate failed attempts.
- **Require CAPTCHA After Failed Attempts** - Require users to successfully identify the contents of a CAPTCHA after a set number of failed login attempts. The prompt for the CAPTCHA will help to protect against automated attacks. One such implementation is reCAPTCHA (https://developers.google.com/recaptcha/docs/start). Note that reCAPTCHA

requires access to the internet.

- **Account Lockout** - If account access must be critically protected and there is a strong business need, the following strategies can be implemented. If these strategies are implemented, attackers could potentially enumerate accounts and intentionally lock valid users out of their own accounts. As such, this approach should be balanced against the requirements of the environment in which pfSense Community Edition Firewall is being deployed.
    - Lock the account from further authentication attempts for a fixed period of time.
    - Lock the account from further authentication attempts and require manual intervention by an administrator to re-enable the account.

In addition to the mitigations listed above, logging all authentication attempts and notifying users login activity via email whenever a suspicious login attempt from an unknown IP or too many failed attempts are made in a short period of time. This would not only help administrators in identifying the root cause of any malicious activity, but would help users in identifying any malicious activity being performed on their account.

## Problem Report 3 - Cross-Site Request Forgery

The pfSense Community Edition Firewall is vulnerable to Cross-Site Request Forgery attacks. An attacker can coerce authenticated users into performing actions without their consent or knowledge of them performing that action.

The Community Edition Firewall adequately protects most POST requests, considered a best practice, from this type of attack. There are, however, several instances in which requests with server side-effects are sent to the application as GET requests and are not protected.

| **Component** | pfSense Web Interface |
|---|---|
| **STRIDE** | Tampering, Denial of Service, Elevation of Privilege |
| **CWE** | CWE-352: Cross-Site Request Forgery (CSRF) |
| **CAPEC** | CAPEC-62: Cross Site Request Forgery (aka Session Riding) |
| **CVSS v2 Score** | 6.1 (AV:N/AC:H/Au:N/C:N/I:P/A:C) |
| **OWASP Reference** | OWASP Top 10 2010-A5: Cross-Site Request Forgery (CSRF) |

| **Overall Severity** | **Medium** |
|---|---|
| **Vulnerability Type** | **Directly Exploitable** |
| **Impact** | **Medium** |
| **Confidentiality** | No information is directly exposed although information could be exposed as a result of action taken as a result of the forged request. |
| **Integrity** | An attacker is able to modify some but not all information contained within the system. |
| **Availability** | An attacker is able to completely deny access to the application and to components behind the router. |
| **Exposure** | The application and users are vulnerable to denial of service attacks. |
| **Affected Users** | All users of the application are affected. |
| **Likelihood** | **Medium** |
| **Skill Required** | An attacker need only be familiar with how HTTP requests work and be able to write basic HTML to craft an attack. |
| **Conditions and Complexity** | An attacker must have knowledge of the address used to access the pfSense Web Interface and be able to coerce a user into visiting an attacker controlled page while logged into the application. Some instances also require further interaction from the user to commit changes or potentially specialized knowledge of the internal configuration. |
| **Discoverability** | This can be easily discovered by users with access to the application. |
| **Reproducibility** | Always reproducible. |

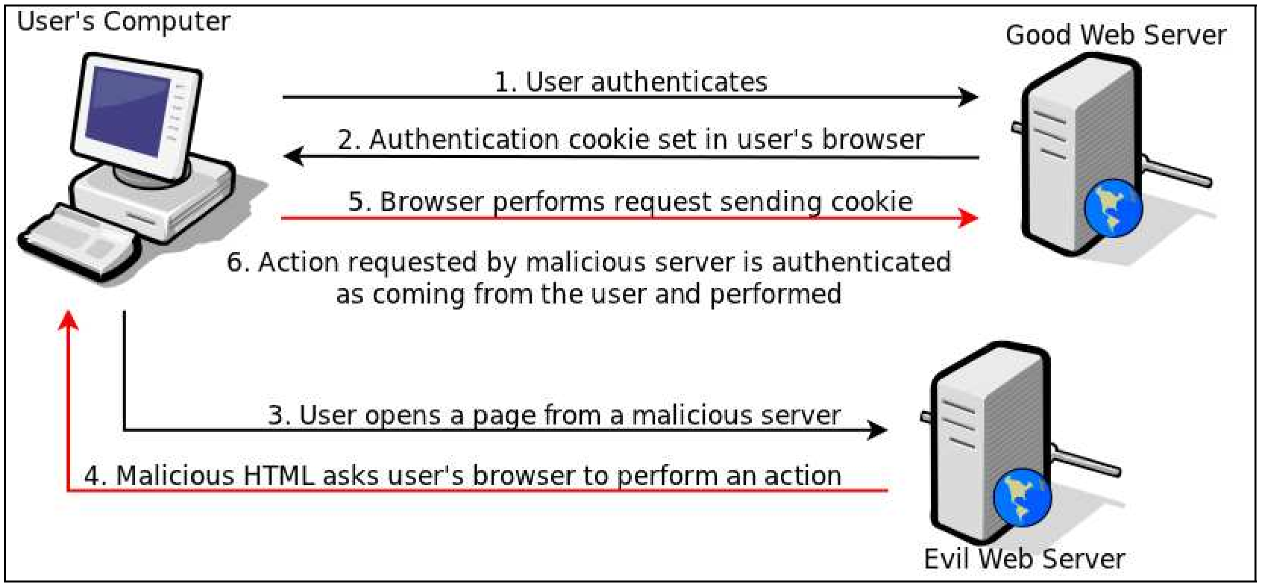
### *Background Information*

The Cross-Site Request Forgery (CSRF) attack (also known as a session riding attack) exploits the design and normal use of browser cookies. Cookies are used to store session identifiers and are automatically included in HTTP requests to a server by the web browser.

The following steps detail what occurs in a general CSRF attack. Everything that happens after the user opens the attacker-controlled page is done in the background, without the user's consent or knowledge.

1. A user authenticates to a vulnerable web application. Typically, this is done with a username and a password.
2. The server gives the user a session ID, which is stored as an HTTP Cookie.
3. The user opens a webpage which contains attacker-controlled HTML or JavaScript. This may be done in a separate tab or window and can come from a completely different server.
4. The malicious HTML or JavaScript silently submits a POST or GET request from the user's browser to the vulnerable server. The attacker can use obscure methods for accomplishing this, such as embedding a GET request in an image tag or a POST request within an IFrame. The browser will submit the request when it renders the webpage.
5. When the malicious request is sent to the vulnerable server, the user's session cookie is automatically sent as well.
6. The server validates the cookie and performs the action on behalf of the unsuspecting user.

Figure 1 visually demonstrates what occurs during a CSRF attack:


User's Computer
Good Web Server
1. User authenticates
2. Authentication cookie set in user's browser
5. Browser performs request sending cookie
6. Action requested by malicious server is authenticated as coming from the user and performed
3. User opens a page from a malicious server
4. Malicious HTML asks user's browser to perform an action
Evil Web Server

**Note:** The attacker does not have access to read the cookie data or server response directly, and the malicious page can be constructed using simple HTML or JavaScript.

By compelling the user to send a request, an attacker can perform any action for which the user is authorized. Additionally, an attacker can chain other vulnerabilities together, such as using a CSRF attack to perform a SQL injection attack on pages the attacker may not have access to. CSRF attacks can be constructed to perform multi-step operations as well.

### Problem Details

In general, pfSense Community Edition Firewall was found to be vulnerable to CSRF when performing delete actions against various assets. The following sample URLs indicate areas of the application which were found to be vulnerable.

This list may not include all CSRF vulnerabilities in the system. It is suggested that, upon remediation, the development team review areas of code where similar techniques are used in order to find and fix similar vulnerabilities.

#### Affected Areas

The following instances immediately delete the indicated resources without any further interaction from the user being necessary:

- `/system_camanager.php?act=del&id=0`
- `/system_certmanager.php?act=del&id=0`
- `/system_routes.php?act=del&id=0`
- `/system_authservers.php?act=del&id=0`
- `/interfaces_groups.php?act=del&id=0`
- `/interfaces_qinq.php?act=del&id=0`
- `/interfaces_ppps.php?act=del&id=0`
- `/interfaces_gre.php?act=del&id=0`
- `/interfaces_gif.php?act=del&id=0`
- `/interfaces_bridge.php?act=del&id=0`
- `/interfaces_lagg.php?act=del&id=0`

The following instances require further user interaction, which prevents the requests from being vulnerable to CSRF attacks:

- `/system_advanced_sysctl.php?act=del&id=0`
- `/system_gateways.php?act=del&id=0`
- `/system_gateway_groups.php?act=del&id=0`
- `/firewall_aliases.php?act=del&tab=ip&id=0`
- `/firewall_aliases.php?act=del&tab=port&id=0`
- `/firewall_aliases.php?act=del&tab=url&id=0`
- `/firewall_aliases.php?act=del&tab=all&id=0`

The following instances require additional knowledge about the existing configuration in order to be exploited:

- `/system_usermanager.php?act=deluser&userid=1&username=test`
  - Requires knowledge of the userid and username
- `/system_groupmanager.php?act=delgroup&groupid=2&groupname=test`
  - Requires knowledge of the groupid and groupname

- `/diag_arp.php?deleteentry=192.168.56.1`
    - Requires knowledge of IPs present in the ARP table
- `/diag_confbak.php?rmver=1490781228`
    - Requires knowledge of the timestamp of any configuration backup.

Security Innovation also observed that a diagnostic ping request could be executed via CSRF:

- `/diag_ping.php?host=test&count=3`

### *Test Steps*

#### Test Configuration

The following is needed in order to reproduce this issue:

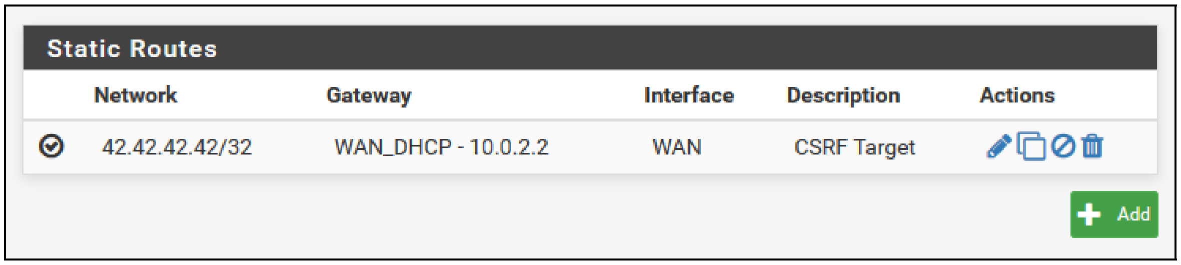
- A running instance of pfSense Community Edition Firewall version 2.3.3-RELEASE-p1
- An account with access to:
    - System / Routing / Static Routes
    - Firewall / Aliases / All

*Note: The following steps can be adapted to reproduce the same issue using any of the vulnerable pages specified in the Problem Details section above.*

#### Steps to Reproduce

**Deleting Static Routes**

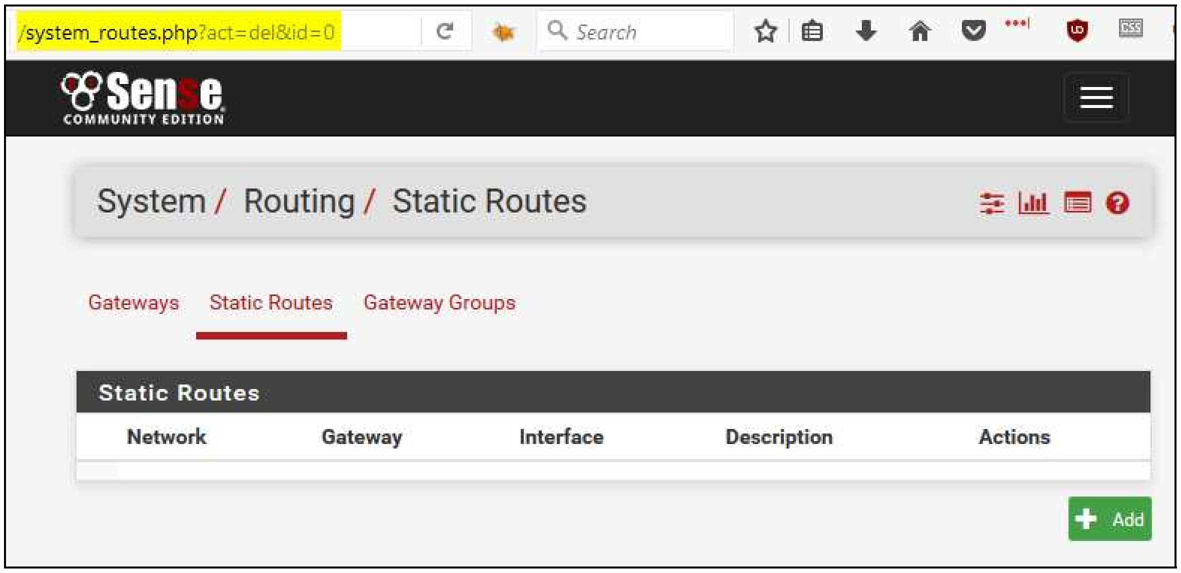
1. Login to the pfSense Web Interface
2. Access the System / Routing / Static Routes page: `/system_routes.php`
3. If no aliases are listed, create one using the `Add` button beneath the table.



Static Routes

| | Network | Gateway | Interface | Description | Actions |
|---|---|---|---|---|---|
| ✓ | 42.42.42.42/32 | WAN_DHCP - 10.0.2.2 | WAN | CSRF Target | |

4. Simulate a CSRF request by accessing `/system_routes.php?act=del&id=\0`
5. Notice that the entry has immediately been deleted.


/system_routes.php?act=del&id=0

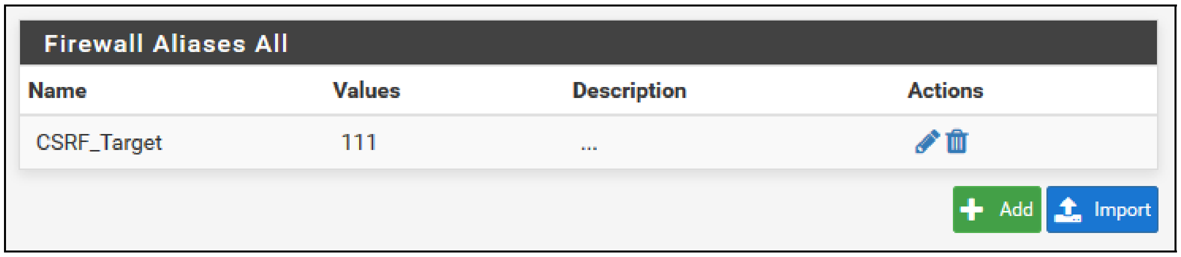
System / Routing / Static Routes

**Static Routes**

| Network | Gateway | Interface | Description | Actions |
|---|---|---|---|---|

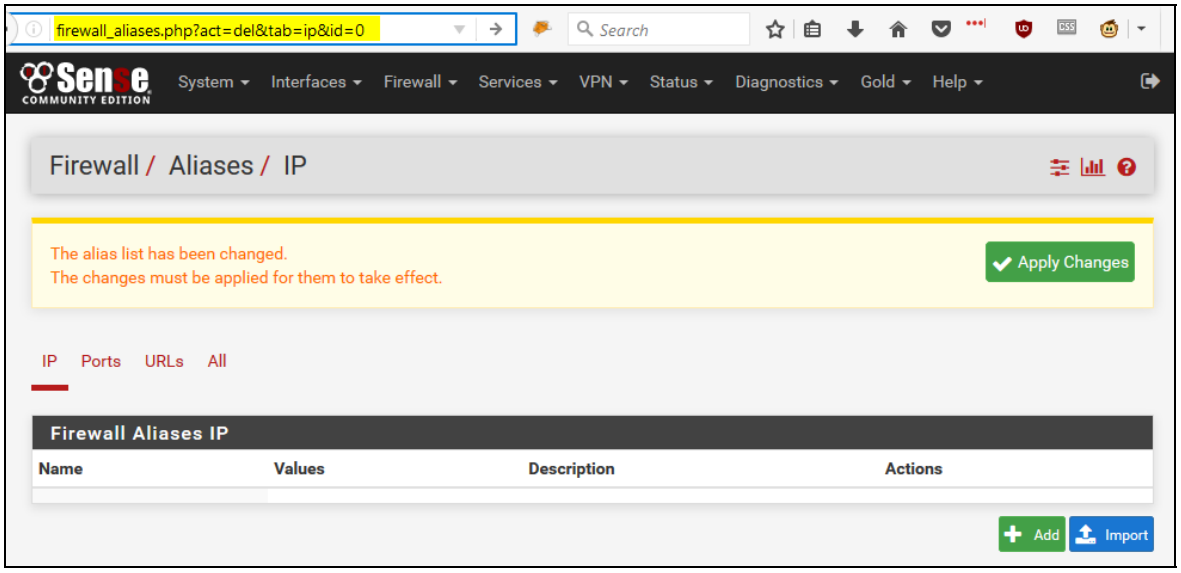
**Deleting Firewall Aliases (Requires further interaction to commit change)**

1. Login to the pfSense Web Interface
2. Access the Firewall Aliases page: /firewall_aliases.php?tab=all
3. If no aliases are listed, create one using the Add button beneath the table.


**Firewall Aliases All**

| Name | Values | Description | Actions |
|---|---|---|---|
| CSRF_Target | 111 | ... | |

4. Simulate a CSRF request by accessing: /firewall_aliases.php?act=del&tab=ip&id=0
5. Notice that the entry has been deleted, but further interaction is still necessary to commit the change.



The alias list has been changed.
The changes must be applied for them to take effect.

## Remediation

**Never perform actions on the server based on GET requests.** If GET requests are used to perform actions on the server, anti-CSRF tokens must be included in these requests. However, these tokens may be leaked in browser history, HTTP log files, Referer headers, and elsewhere. In addition, certain CSRF-prevention frameworks (for example, *protect_from_forgery* in Ruby on Rails) only verify CSRF tokens present in POST requests. For these reasons, it is important to only include anti-CSRF tokens in HTTP POST bodies and never perform user actions based on GET requests.

**Use the csrf-magic token for all requests with server side-effects.** Enumerate all HTTP requests that case server-side changes and ensure that the csrf-magic library is used to protect those endpoints.

**Require users to re-authenticate or provide a One-Time Password or PIN in the form of Two-Factor Authentication (2FA) before performing dangerous operations.** Whether this measure is necessary depends on the sensitivity of the action being performed. This will require the user to enter their password or a 2FA token for operations such as adding users or changing passwords. The 2FA token may be sent to the user via text, email, or another method.

## Additional Resources

Many languages and frameworks provide built-in CSRF Synchronizer Token Pattern protections. Platform specific details can be found below:

### PHP

PHP does not provide built-in CSRF protection, however, the csrf-magic library dynamically rewrites HTML forms and AJAX requests to include anti-CSRF tokens, as currently used by Community Edition Firewall .

- http://csrf.htmlpurifier.org

- https://www.owasp.org/index.php/PHP_CSRF_Guard

## General References

- OWASP CSRF Overview
    - http://www.owasp.org/index.php/CSRF
- OWASP CSRF Prevention Cheat Sheet

  http://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

- The Cross-Site Request Forgery (CSRF) FAQ

  http://www.cgisecurity.com/csrf-faq.html

- TeamMentor CSRF

  https://services.teammentor.net/teamMentor#load:22dc0732-ec5e-4600-9a1c-97d0f83545f7

## Problem Report 4 - Cross-Site Scripting (XSS)

The Community Edition Firewall application is vulnerable to Cross-Site Scripting (XSS). An attacker can run malicious scripts in the context of the website in order to steal a user's cookies and impersonate the user, coerce the user into carrying out unintended actions on the web application, or modify the content displayed to the user.

| | |
|---|---|
| **Component** | pfSense Web Interface |
| **STRIDE** | Spoofing, Tampering, Information Disclosure, Denial of Service, Elevation of Privilege |
| **CWE** | CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') |
| **CAPEC** | CAPEC-32: Embedding Scripts in HTTP Query Strings |
| **CVSS v2 Score** | 6.0 (AV:N/AC:M/Au:S/C:P/I:P/A:P) |
| **OWASP Reference** | OWASP Top 10 2013-A3: Cross-Site Scripting (XSS) |

| | |
|---|---|
| **Overall Severity** | **High** |
| **Vulnerability Type** | **Directly Exploitable** |
| **Impact** | **High** |
| **Confidentiality** | All data available to the user may be compromised. |
| **Integrity** | Integrity of data is not immediately compromised but compromise of an administrative user would result in potential compromise of all system files. |
| **Availability** | An attacker could completely shutdown the system if they compromise an administrative user. |
| **Exposure** | Any anonymous actor could compromise privileged users of the system. |
| **Affected Users** | All users with access to the vulnerable pages are vulnerable. |
| **Likelihood** | **Medium** |
| **Skill Required** | An attacker would require a moderate understanding of how cross-site scripting works to exploit these attacks. |
| **Conditions and Complexity** | An attacker would need to already have access to the system or have knowledge of the pfSense Web Interface URL. |
| **Discoverability** | An attacker would require moderate knowledge of cross-site scripting attacks to discover these issues. |
| **Reproducibility** | The attack is always reproducible. |

### *Background Information*

**Impact**

A Cross-Site Scripting (XSS) attack can cause arbitrary code to run in a user's browser in the context of a trusted website. The attack targets your application's users (not the application itself) and uses your application as the vehicle for the attack.

Because the script code is downloaded by the browser from a trusted site, the browser has no way of knowing that the code is not legitimate. As the attacker's code has access to cookies scoped to the trusted site and that are stored on the user's local computer, a user's authentication cookies are typically the target of attack. Additionally, maliciously injected JavaScript running in a user's browser can also access the current URL, any element of the DOM, and even port scan the local network.

**Types of XSS**

There are two primary types of XSS, *persistent and non-persistent*.

If the attacker is able to inject their malicious script into an area of the web applications that stores data, the script will persist and anyone who then loads a page with that content will run the script and become a victim of the attack. Forums and other web services that have user supplied HTML content are a classic example of this. If this content is not validated and encoded, then anyone can leave a malicious script on the site.

Non-Persistent XSS uses some reflective aspect of the page to deliver the payload. Web applications often echo user input back to them without altering it. Custom search functionality is a good example of a feature that can reflect user input. Attackers can use these reflection points to create and distribute URLs that contain a malicious scripts that get reflected back to the user. Another common example of this is when a 404 error page echoes the requested page back to the user.

**Exploit Scenarios**

The following examples are included to demonstrate different exploit scenarios and the fact that there are always multiple ways to bypass weak blacklist based security measures. The most common XSS exploit uses HTML script tags to trigger code execution in the user's browsers. When user input, persistent or reflected, is rendered by the browser without rigorous whitelist validation, code can be included in a executable format. The following string is a classic example

```
<script>alert("xss");</script>
```

In this simple case, the exploit can be prevented by disallowing the `<` and `>` symbols using proper HTML encoding of output. The following safe equivalent is displayed as text on the webpage instead.

```
&LT;script&GT;alert(&QUOT;xss&QUOT;);&LT;/script&GT;
```

Additionally, user input is often included in JavaScript directly as string literals in variable assignments, as HTML entity attribute values, or placed in the DOM dynamically. Consider the following scenarios:

```
<img src="gallery/kitten.png" alt="kitten" />
```

If the application accepts user input for the *alt* attribute, an attacker can provide the following image alt tag data instead, thus resulting in JavaScript execution without using the `<` or `>`

symbols:

```
alt" onmouseover="alert('xss')" blah="
```

In certain cases the same thing can be accomplished without quotations by leveraging the DOM.

### *Problem Details*

The following areas of the application were found to be vulnerable. This list may not include all instances of this vulnerability in the system. It is suggested that upon remediation, the development team review other areas of the Community Edition Firewall where similar techniques are used in order to find and fix related vulnerabilities.

**Diagnostics / Edit File**

The file browser used by the Community Edition Firewall is vulnerable to a persistent cross-site scripting attack. This attack stems from the lack of output encoding or sanitization of the system filenames.

The relevant source code from *vendor/filebrowser/browser.php* is included below:

**vendor/filebrowser/browser.php**

```
<?php $filename = str_replace("//","/", "{$path}/{$file}"); ?>
<div onClick="$('#fbTarget').val('<?=$filename?>'); loadFile();
$('#fbBrowser').fadeOut();">
```

**Firewall / NAT / Port Forward / Edit**

The Firewall NAT editor is vulnerable to a persistent cross-site scripting attack. The interface name is encoded by _htmlspecialchars_. However, this is not sufficient as the _htmlspecialchars_ function does not encode single quotes allowing an attacker to escape the existing JavaScript and inject their own by crafting their own interface name. A successful attack utilizing this vulnerability would require an internal attacker with access to create or modify NAT rules.

The relevant source code from *firewall_nat_edit.php* is included below:

**firewall_nat_edit.php**

```
dst_change($('#interface').val(),'<?=htmlspecialchars($pconfig['interface'])?
>','<?=htmlspecialchars($pconfig['dst'])?>');
```

**Diagnostics / Tables**

The Diagnostics page displays a list of address tables and allows you to select a table to display. This table is specified in the user-controlled URL parameter type (referenced in source as $tablename) that is subsequently parsed into some JavaScript. Similar to the previous issue, it is wrapped with single quotes and output using _htmlspecialchars_, allowing an attacker to escape the string and inject their own JavaScript. This attack can be performed by an external attacker against any legitimate user with access to the Tables page and has knowledge of the Community Edition Firewall url.

The relevant source code from *diag_tables.php* is included below:

**diag_tables.php**

```
$.ajax(
    '/diag_tables.php',
    {
        type: 'post',
        data: {
            type: '<?=htmlspecialchars($tablename)?>',
            delete: $(this).data('entry')
        },
        success: function() {
            el.parents('tr').remove();
        },
});
```

### *Test Steps*

#### Test Configuration

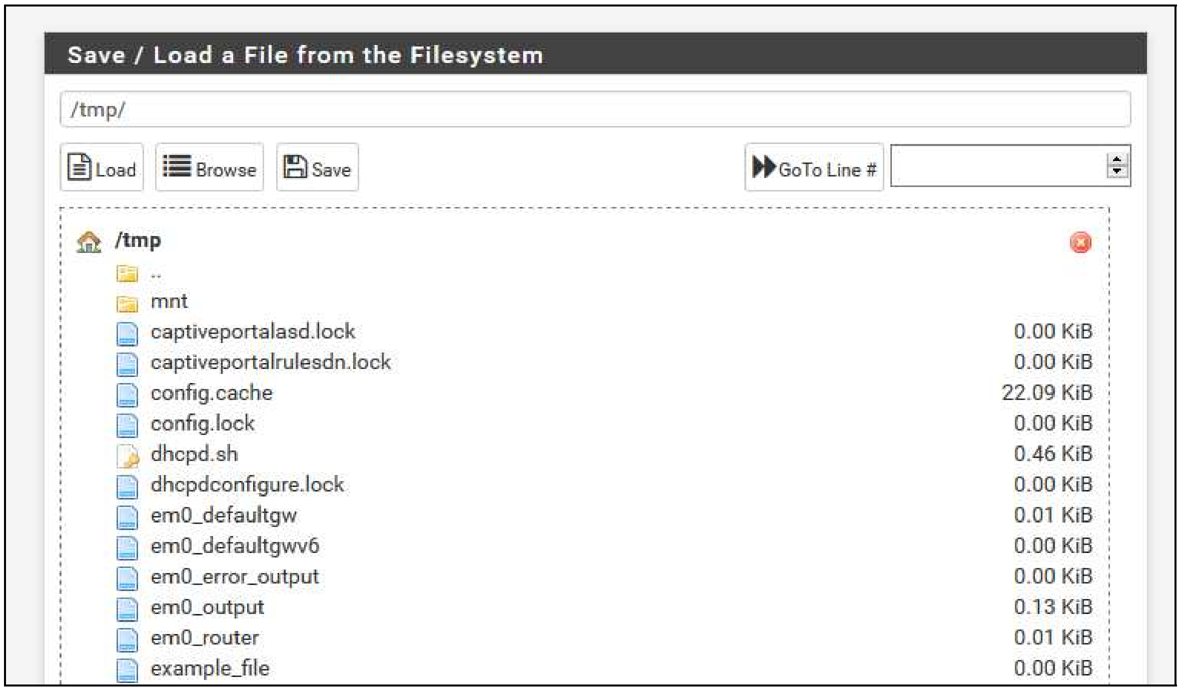
The following is needed in order to reproduce this issue:

- A running instance of pfSense Community Edition Firewall version 2.3.3-RELEASE-p1
- An account with access to:
    - Diagnostics / Edit File
    - Firewall / NAT / Port Forward / Edit
    - Diagnostics / Tables

#### Steps to Reproduce

**Diagnostics / Edit File**

1. Login to the Community Edition Firewall pfSense Web Interface
2. Navigate to Diagnostics / Edit File
3. Load any file

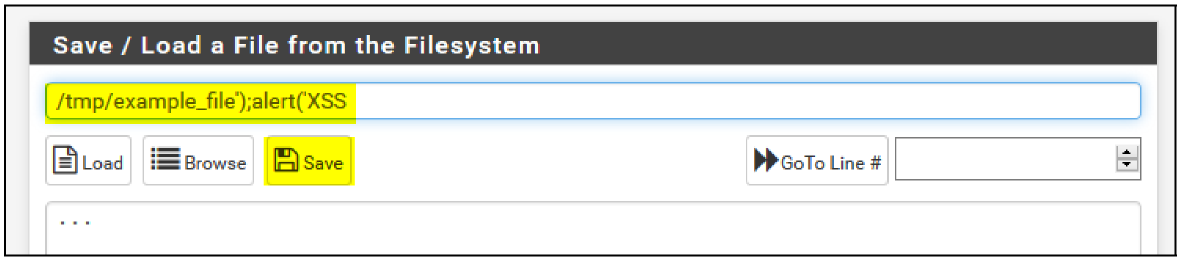


Save / Load a File from the Filesystem

/tmp/

| | |
|---|---|
| /tmp | |
| .. | |
| mnt | |
| captiveportalasd.lock | 0.00 KiB |
| captiveportalrulesdn.lock | 0.00 KiB |
| config.cache | 22.09 KiB |
| config.lock | 0.00 KiB |
| dhcpd.sh | 0.46 KiB |
| dhcpdconfigure.lock | 0.00 KiB |
| em0_defaultgw | 0.01 KiB |
| em0_defaultgwv6 | 0.00 KiB |
| em0_error_output | 0.00 KiB |
| em0_output | 0.13 KiB |
| em0_router | 0.01 KiB |
| example_file | 0.00 KiB |

4. Append the following snippet to end of the populated filename as shown below and press Save to rename the file: `');alert('XSS`

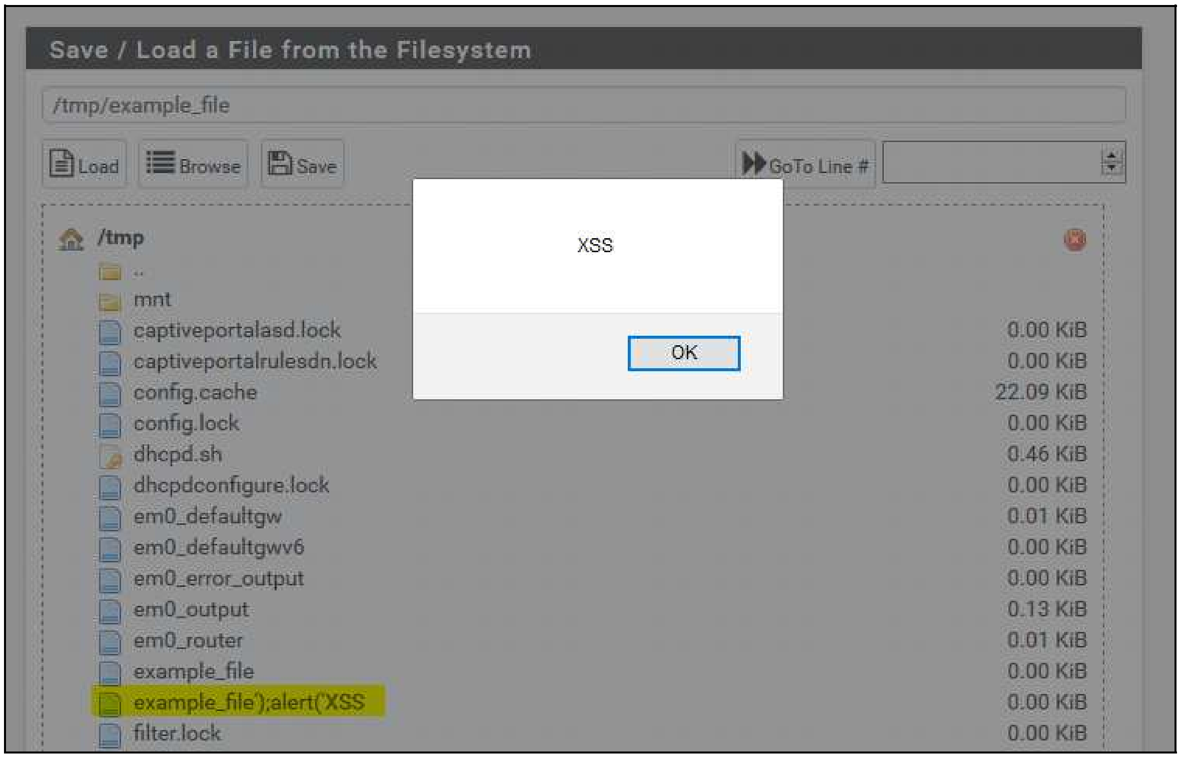


Save / Load a File from the Filesystem

/tmp/example_file');alert('XSS

5. Click Browse to display the file list again

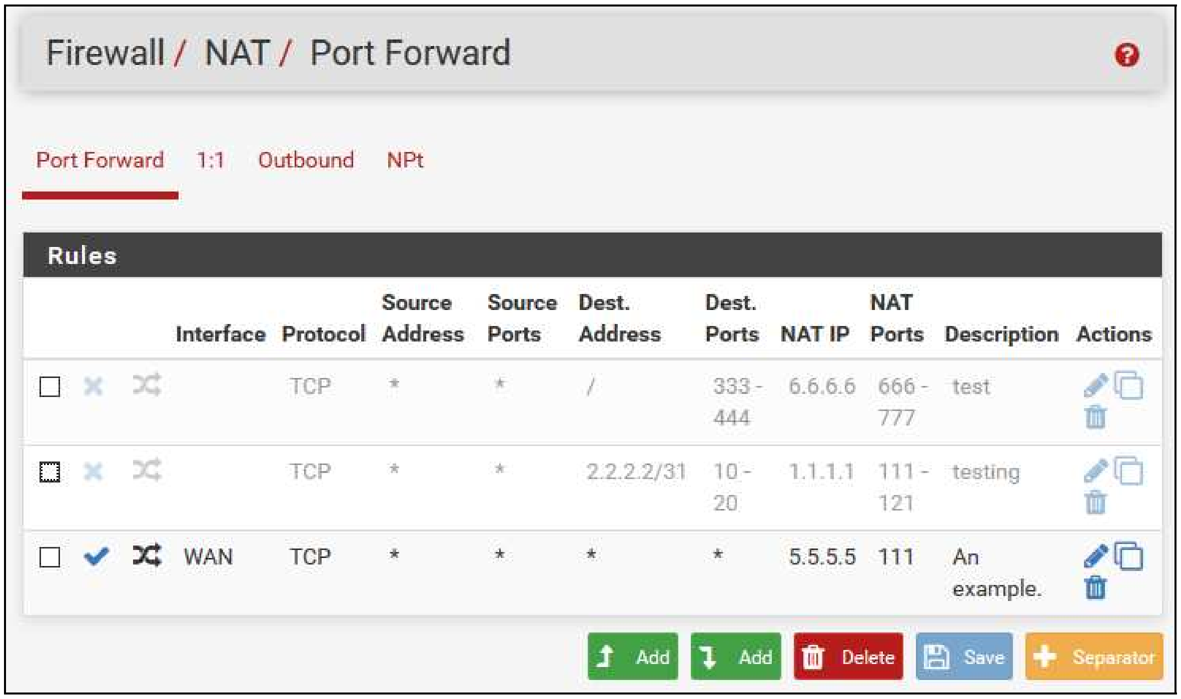
6. Click the newly created filename to execute the injected script.



```
..
mnt
captiveportalasd.lock          0.00 KiB
captiveportalrulesdn.lock      0.00 KiB
config.cache                  22.09 KiB
config.lock                    0.00 KiB
dhcpd.sh                       0.46 KiB
dhcpdconfigure.lock            0.00 KiB
em0_defaultgw                  0.01 KiB
em0_defaultgwv6                0.00 KiB
em0_error_output               0.00 KiB
em0_output                     0.13 KiB
em0_router                     0.01 KiB
example_file                   0.00 KiB
example_file');alert('XSS      0.00 KiB
filter.lock                    0.00 KiB
```

**Firewall / NAT / Port Forward / Edit**

1. Login to the Community Edition Firewall pfSense Web Interface
2. Navigate to Firewall / NAT / Port Forward



Firewall / NAT / Port Forward

Rules

| | | | Interface | Protocol | Source Address | Source Ports | Dest. Address | Dest. Ports | NAT IP | NAT Ports | Description | Actions |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ☐ | ✗ | | | TCP | * | * | / | 333 - 444 | 6.6.6.6 | 666 - 777 | test | |
| ☐ | ✗ | | | TCP | * | * | 2.2.2.2/31 | 10 - 20 | 1.1.1.1 | 111 - 121 | testing | |
| ☐ | ✓ | | WAN | TCP | * | * | * | * | 5.5.5.5 | 111 | An example. | |

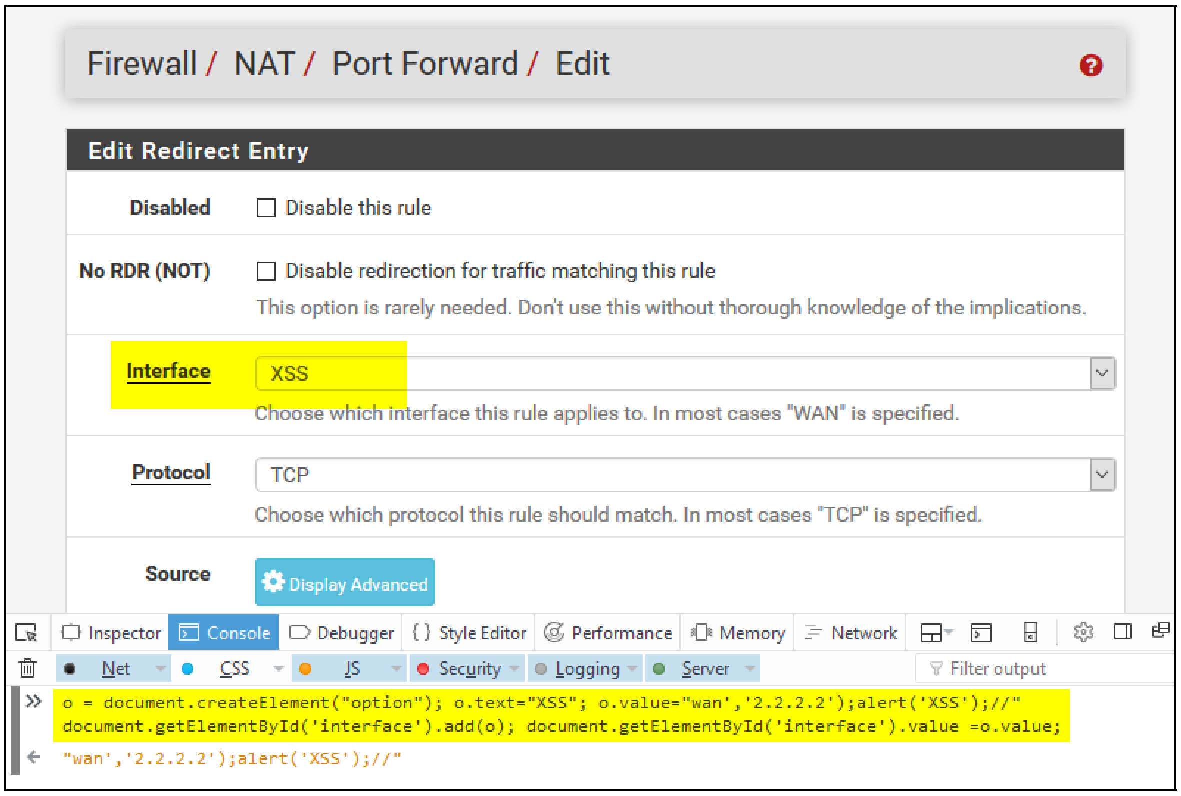
3. Select Edit for any existing element

4. Open your browser's Web Developer Console

   - Firefox: Ctrl+Shift+K
   - Chrome: F12

5. Run the following javascript in the console as shown below and save the form:

```
o = document.createElement("option");
o.text="XSS";
o.value="wan','2.2.2.2');alert('XSS');//"
document.getElementById('interface').add(o);
document.getElementById('interface').value =o.value;
```


Firewall / NAT / Port Forward / Edit

Edit Redirect Entry

Disabled ☐ Disable this rule

No RDR (NOT) ☐ Disable redirection for traffic matching this rule
This option is rarely needed. Don't use this without thorough knowledge of the implications.

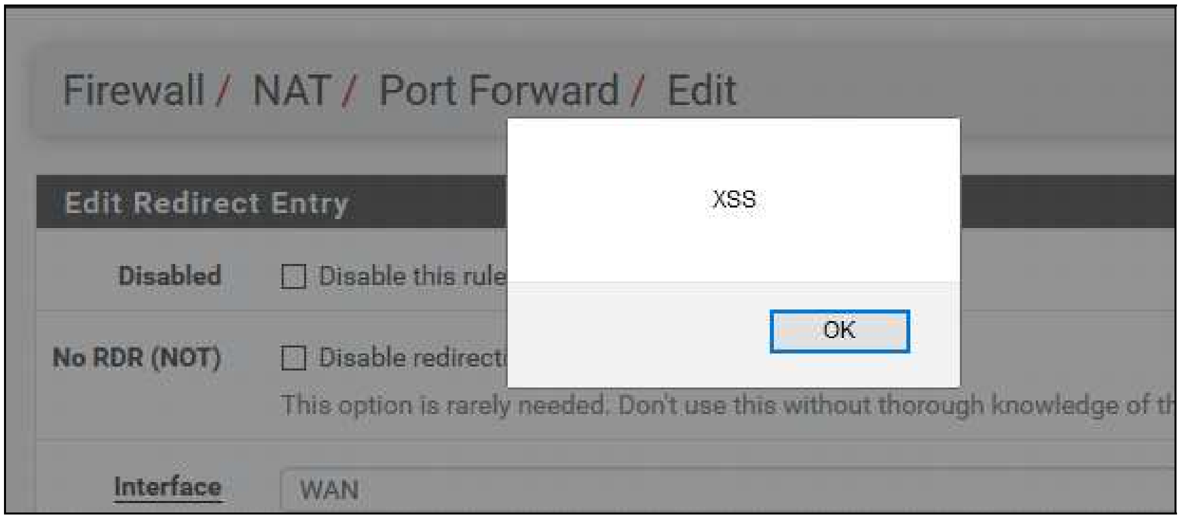
Interface XSS
Choose which interface this rule applies to. In most cases "WAN" is specified.

Protocol TCP
Choose which protocol this rule should match. In most cases "TCP" is specified.

Source Display Advanced

```
o = document.createElement("option"); o.text="XSS"; o.value="wan','2.2.2.2');alert('XSS');//"
document.getElementById('interface').add(o); document.getElementById('interface').value =o.value;
"wan','2.2.2.2');alert('XSS');//"
```

6. Attempt to edit the modified rule again and the injected code will execute


Firewall / NAT / Port Forward / Edit

Edit Redirect Entry

Disabled ☐ Disable this rule

No RDR (NOT) ☐ Disable redirect

This option is rarely needed. Don't use this without thorough knowledge of th

Interface WAN

XSS

OK

**Diagnostics / Tables**

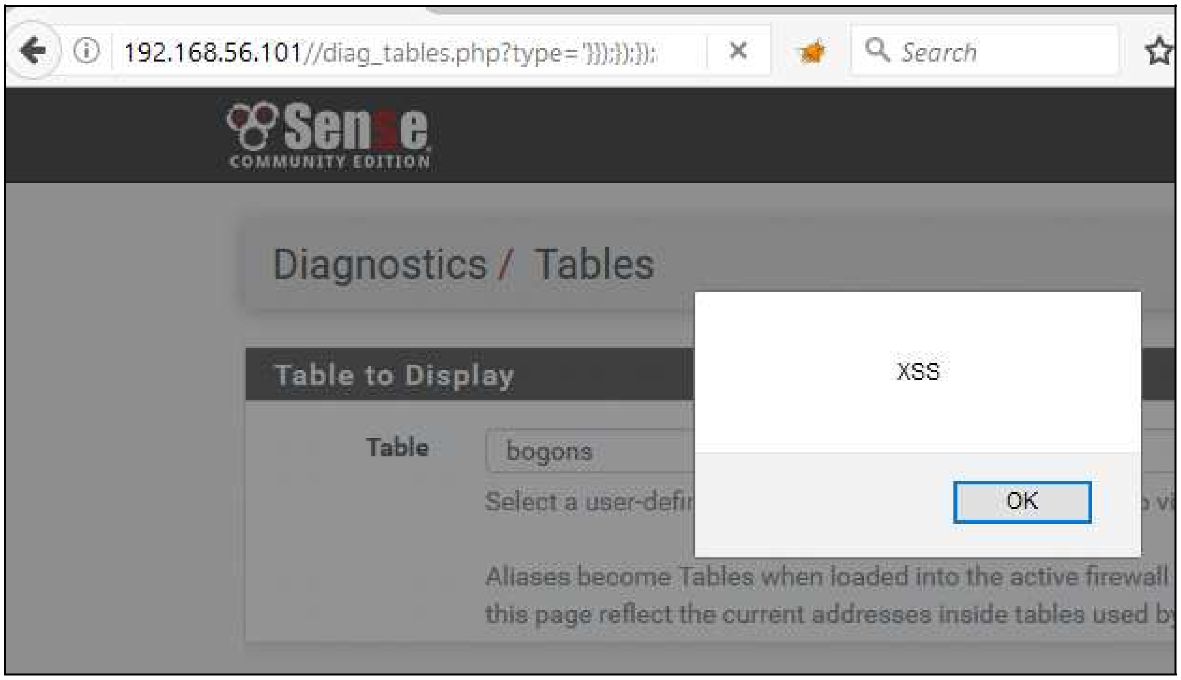
1. Login to the Community Edition Firewall pfSense Web Interface

2. Access the following address:

```
/diag_tables.php?type=%27}});});});alert(%27XSS%27);%20console.log(function()
```

```
{%20console.log(function(){%20console.log({%20c:{%27a%27:%27
```

3. Notice the alert that pops up indicating a successful attack.


192.168.56.101//diag_tables.php?type='}});});
Search
Sense
COMMUNITY EDITION
Diagnostics / Tables
Table to Display
Table
bogons
Select a user-defi
XSS
OK
Aliases become Tables when loaded into the active firewall
this page reflect the current addresses inside tables used b

### *Remediation*

**Use proper output encoding.** Whenever untrusted data is reflected into resulting HTML, it must be encoded to avoid XSS. Output encodings must be context specific, encoding HTML, attributes, and URLs with their proper encoding methods. Additionally, trusted encoding methods provided by languages or frameworks must always be preferred over custom implementations.

**Validate all user input using rigorous whitelist style checking on the server.** Regular expressions, inclusion lists, casting of primitive types, range constraints, and other means can be used to ensure that all user input contains only acceptable characters and has an appropriate format. Especially with web applications, there are many ways to represent the same data, so no blacklist is ever sufficient.

Validation must always be done on the server. Client-side validation may be used to provide a good user experience, but can always be bypassed and hence, is not useful as a security measure. Note that most HTML encoding libraries only encode `<`, `>`, and `"`.

**Use context sensitive validation routines.** When user input is supposed to be a phone number, only accept ten numerical digits and strip all other characters. When the user is inputting an email address, validate that only alphanumeric, periods `.`, and a single at-symbol `@` are present. Reject any input that does not fit the whitelist template for that data type. Consider the context of the input and whitelist appropriately. If special characters, such as quotations, are necessary, verify that the data is securely encoded throughout all components of the application.

**Use double quotations for HTML attributes.** When HTML entity encoding is performed on the server for user input, it is important that all HTML entities are consistently enclosed in double `"` instead of single `'` quotes because many HTML encoding methods do not encode apostrophes (single quotes). Consider the following example HTML:

```
<input name="email_address" id="email" value=' ' type="text">
```

When user input is reflected from the server into the "value" attribute, a malicious attacker can include single quotes to alter the meaning. In particular, this input can include JavaScript keywords such as "onfocus" or "onmouseover" to embed malicious code.

## Follow-up

Inconsistencies, errors and reproducibility problems associated with this report should be directed through the customer contact person to the tester and preparer indicated at the beginning of this report.