# Title: Security Best Practices

|  |  |
| --- | --- |
| Weakness | Recommendation |
| SSRF | SSRF helps an attacker to send a crafted request from a vulnerable web application to access internal systems behind firewalls that are normally inaccessible to the public. In our application, we were able to do the same via **HTTP request proxy** to access internal servers which are hosted in AWS. We believe that most of our applications are using HTTP request proxy to check the URL or to get data from the integration / third-party application.  1. Blacklist all private IP's and loopback addresses     · [10.0.0.0/8](http://10.0.0.0/8)     · [172.16.0.0/12](http://172.16.0.0/12)     · [192.168.0.0/16](http://192.168.0.0/16)     · Link-local unicast: [169.254.0.0/16](http://169.254.0.0/16) & fe80::/10     · Loopback interface: [127.0.0.0/8](http://127.0.0.0/8) & ffx1::/16 2. Disable protocols such as <file:///>, dict://, ftp:// and gopher:// that helps attacker to access our server files (Mostly anything apart from https://). 3. Disable following of HTTP redirects that are happening via HTTP request proxy.  Reference Tickets: [FS-5060](https://freshworks.freshrelease.com/FS/issues/FS-5060) and [FSALES-3948](https://freshworks.freshrelease.com/FS/issues/FSALES-3948) |
| safe\_send | [send](https://ruby-doc.org/core-2.5.0/Object.html#method-i-send) is a ruby method that allows invoking another method by name. It is used as a common way of calling methods dynamically ( when we don't know the method name in advance )  **send("method\_name",args)**  Vulnerable code snippet:  method = params [ :method ] @result = User.send(method.to\_sym)  In the above snippet, the value of "method" parameter is controlled by the user. Direct user input must not be passed onto the send method. For eg, an attacker could invoke the 'system()' method to execute system commands.  **send('system','ls') -> Executes 'ls' command.**  Remediation:  1. Whitelist the methods that you are going to call using the send method.  2. Freshdesk product team has coded "**safe\_send**" method which overrides the send method. safe\_send blacklists some of the riskier methods such as system, eval, instance\_eval, send, \_send\_, public\_send, exit etc.., We could use **safe\_send**instead of send.  safe\_send code information: <https://github.com/freshdesk/helpkit/blob/dfacb5bda7b67bf6452a27c394b7ebf7de0a60bd/lib/helpdesk/initializers/core_ext/object.rb>  3. Brakeman also flags this issue as "[**Dangerous send**](https://brakemanscanner.org/docs/warning_types/dangerous_send/)". We could check for "Dangerous send" warning in brakeman results.  <https://docs.google.com/presentation/d/13CnQy0xjb5cb3f5aAL7BpH0anxh0BwDoSxUs_GytiJk/edit?usp=sharing> |
| SSO | Protecting Customer data is the top priority of Freshworks. Whenever a customer faces any issue in our products, a ticket is raised to our support team. In order to debug the issue in depth, our agents/engineers might require access to the customer’s portal.  Why Access is required :   To validate, debug or confirm the occurrence of the issue reported by the customer.  How do we get access :   1. Ask the customer to add our agent(s) as an occasional agent(s) and login with a day pass. 2. Single Sign-on is used under unavoidable situations.     Although the intent is to debug the issue, viewing customer’s data should happen ONLY with their consent. Regardless of the fact that only a customer can create an occasional agent for us, an intimation must be given to the customer every time we need to log into their portal.  Process change recommendation :   * The support agent addressing the issue should intimate the customer every time a login is required to the portal. As a temporary remediation, a ticket conversation which illustrates the ‘Purpose of login’ should be sent to the customer everytime a login is to be made. * Reusing the same password for every occasional agent, across portals is discouraged. * SSO must be enabled or disabled only by the customer, or if any configuration help on SSO is required, it must be altered ONLY after getting a consent for the same.     Product change recommendation :   * Customer controlling the occasional agent's expiry : In this case, a customer can create an occasional agent with an expiry time (24 hours or 48 hours or 1 week), post which a login cannot be made and any existing session will timeout. Additionally, an email can be sent to the customer every time a login is made within the timeframe. * In case of Single Sign on: The url generated must be usable only once and must be valid only for ±30 seconds. * Whenever customer disables SSO, the shared secret must be regenerated. * As a long term solution, SSO should be managed by a central framework across all products, in our case, FreshID. |
| AWS Config | AWS Config is a service that enables us to audit and evaluate the configurations of the AWS resources.  This service uses AWS CloudTrail data for operating and producing results.  **Primary Features of AWS Config:**   1. Maintains the inventory of supported AWS resources. (Eg., List of security groups present within a region can be viewed).     2. Keeps track of relationship between a resource to other AWS resources (Eg., Security group associated to network interface, EC2 instance and the particular VPC can be seen)      3. Highlights the changes made to the resources (Infrastructure Changes), over a period of time.  4. Project the timeline occurred in past for an AWS resource with exact date and time.      **AWS Config Rules:**  AWS Config rules helps us to perform compliance auditing, monitor on change management process and to perform security analysis to look for insider threats and many. To know more about the AWS config rules, please find the link below:  <https://aws.amazon.com/blogs/aws/aws-config-rules-dynamic-compliance-checking-for-cloud-resources/>  To get more information on AWS config and its supported resources, config snapshots, pricing and FAQ’s please find the links below:  <https://aws.amazon.com/config/>  <https://aws.amazon.com/config/pricing/>  <https://aws.amazon.com/config/faq/> |
| Security Headers | Each header mentioned below has its own advantage, detailed information and its options are available [here](https://docs.google.com/document/d/1Q3qs-7FgEqHmcYnArXMDFmFEu1U4K3qkKy4KArKKGxg/edit).   * **X-Frame-Options** - The X-Frame-Options HTTP response header can be used to instruct the browser whether to render a page in a <frame>, <iframe> or <object> . Web applications can use this to avoid clickjacking attacks, by ensuring that their content is not embedded into other sites. * **X-XSS-Protection** - The HTTP X-XSS-Protection response header is a header that instructs the browser to stops page loading when they detect reflected cross-site scripting attacks. Although these protections are largely unnecessary in modern browsers when sites implement a strong Content-Security-Policy that disables the use of inline JavaScript ('unsafe-inline'), they can still provide protection for users of older web browsers that don't yet support CSP. Their directives for X-XSS-protection are, * **Content Security Policy** - Content Security Policy (CSP) is a header to add a layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks (These attacks are used for data theft from the browser, site defacement, distribution of malware) . To enable CSP, you need to configure your web server to return the Content-Security-Policy HTTP header. * **X-Content-Type-Options** - The X-Content-Type-Options header is used to protect against MIME sniffing vulnerabilities. These vulnerabilities can occur when a website allows users to upload content to a website however the user disguises a particular file type as something else. This can give them the opportunity to perform cross-site scripting and compromise the website. However, this security header helps prevent these types of attacks by disabling the MIME sniffing functionality of IE and Chrome browsers so that the browser is required to use the MIME type sent via the origin server. Consider the following example of how X-Content-Type-Options works for a particular web request. * **Strict-Transport-Security** - The HTTP Strict-Transport-Security response header lets a website forces browsers that it should only be accessed using HTTPS, instead of using HTTP. * **Content-Disposition** - This header can be used if you want all your attachments/user uploaded contents to the website to be downloaded when opened. This could prevent attacks like Cross-site scripting through user-uploaded contents * **Referrer-Policy** - The Referrer-Policy HTTP header governs which referrer information, sent in the Referer header, should be included with requests made.This header is really useful in pages like password-rest, where the URL will contain some secret token. The secret token will be exposed to all the services loading from that page through referrer header. Adding strict Referrer-Policy for those pages will prevent leaking of sensitive tokens in URLs through referrer |
| TLS Encryption | **TLS Encryption:**  **Disadvantages of TLSv1.0 and TLSv1.1:**  The applications using TLSv1.0 and TLSv1.1 are prone to known man-in-the-middle ("MitM") attacks, including the [BEAST](https://bugskan.com/tutorials/owasp-tutorial-testing-for-weak-ssl-tls-cipher/) and [POODLE](https://whatis.techtarget.com/definition/POODLE-attack) attacks.  **Disadvantage of TLSv1.2:**  "[Transcript Collision](https://access.redhat.com/articles/2112261)" is an attack identified on hash functions in cryptographic protocols such as TLS, which can force a hash-construction downgrade to MD5 and reduce expected security. These attacks rely on the use of obsolete hash constructions such as MD5 in these protocols.  **Latest Version:**  **TLSv1.3 - Advantage:**   * All handshake messages after the ServerHello are now encrypted. * The handshake state machine has been restructured to be more consistent and remove superfluous messages. * TLS 1.2 version negotiation verification mechanism was deprecated. * The encryption process in TLSv1.3 also starts with a client hello, but as an added advantage the client also guessed the key algorithm that the server might choose and sends a key share for that as well. The following process is same as that of TLSv1.3, however a round trip is cut short which minimises the process time.   **Recommendation:**  It is recommended to upgrade to TLSv1.3 in all the products(both client side and server side) in order to avoid known vulnerabilities and secure communication between client and server.  **Browser Recommendation:**   * Chrome 65 and Firefox 52 currently support draft version of TLSv1.3. The original versions on both of these browsers are scheduled for release by october, 2018. * Safari supports TLS 1.3 in version 11.1 on macOS High Sierra but it’s disabled by default. This can be enabled manually. * Microsoft Edge, IE and Opera are some of the browsers which do not support TLSv1.3. |
| CSRF | **CSRF - Cross Site Request Forgery:**  CSRF attack occurs when an attacker can craft his actions on target victim in the form of specially-crafted image tags, hidden forms, and JavaScript XMLHttpRequests to trick victim to make any actions such as "clicking on links" and forcing him to process attacker crafted request in victim authenticated browser.  Alias: One-Click Attack Session Riding Sea Surf Attack  **Impact:**  The impact of successful CSRF attack can be vulnerable as the attacker can compromise targeted user’s active session account without knowledge of the victim.  Example: Adding/Removing Agents, Account Deactivation/Deletion,Assigning Agents to Tickets, Creating Solution Articles, Contact Conversions, Upgrading Plans/Billings etc,.  **Remediation / Mitigation:**  The general remediation for preventing CSRF attack is implementing a CSRF token in Post method/Request header for any state changing actions  Below are the list of best practices to be followed for a secure implementation:  Avoid GET method for state changing actions CSRF tokens should not be passed in GET requests - because they can be leaked at several locations like browser history, log files and network appliances. Attacker can reuse the active CSRF token to launch the successful attack.  Validate Token reuse CSRF token should be invalidated once the user active session is closed by the User and all unused/invalid tokens should not be allowed for reuse. A new random token should be generated for each user session. When you build your own CSRF token then below factors always to be present in code CSRF Token = random(active\_session\_id,time\_stamp)  Limit expiry of tokens CSRF tokens should have limited time of expiry and it should be invalidated when the User session is closed.  Validate modified CSRF tokens: Sometimes it is possible to bypass CSRF protections by simply sending blank tokens or modifying some characters of the token to match the length of the token. Hence validation should be in place to prevent bypass.  Validate shared Tokens: There can be chances that two valid tokens can be exchanged or shared between multiple accounts and by using this case an attacker can multiple valid account to perform malicious actions.  Checking the referer header: In many cases it is best practices to check the referer / origin header to verify the request is coming from the trusted origin. But there are case where this can be bypassed by data URI - Example : data:text/html, <script>top.location.href='[http://www.google.com/';</script](http://www.google.com/)>  When this is loaded in browser it will redirect with “no referer” and origin headers as “null”. Hence it is recommend to block request if any of the headers are not present.  Using .swf files It's possible to send the "X-Requested-With" header with Flash, but we can't send it cross-domain due to the SOP implementation with Adobe's crossdomain.xml files. However, we can trick Flash sending the request anyway by issuing the request to a domain that does allow it, which in turn sends a HTTP 307 redirect to the resource that we want to target. Flash will send the request to the new resource before requesting crossdomain.xml on the new domain.  References: [CSRF Cheatsheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet)  Conclusion: All the above recommendations must be considered while implementing the fix for CSRF attacks in web application. |
| Github | List of security best practices to be followed while working with GitHub:  **Refining Sensitive Data into Github:**   * It's always recommended to scan git commit for sensitive data such as AWS Secret Key, Hardcoded FreshID credentials, Server Credentials such as SMTP etc., before  being pushed to GitHub repositories (both Public & Private) by [git-secrets](https://github.com/awslabs/git-secrets) (git pre-commit hook)   **Purging Sensitive Data from Github:**  If any sensitive data is missed due to human-error and made available to git repos (Both Public & Private) then follow the below given steps:   * Invalidate and rotate leaked tokens/secrets/credentials and then create a new commit after removing the secret * Assess impact of leaked private tokens/secrets/passwords [RCA]   **Implementing Tight Controlled Access:**   * Implement two-factor authentication for all your GitHub accounts * Never let users share GitHub accounts/passwords * Any laptops/devices with access to source code must be properly secured * Diligently revoke access from users who are no longer working with Freshworks * Manage team member’s access to data. Give contributors only access to what they need to do their work   **Adding** [**SECURITY.md**](http://SECURITY.md) **file:**   * [SECURITY.md](http://SECURITY.md) file should portray GitHub Security Best Practices of Freshworks which will help developers to understand their responsibilities to prevent exposing sensitive data   **GitHub Apps:**   * Users trying to authorise new apps should send approval request with error message to group mail  security-approvals@freshworks.com     **SSH keys and Personal Access Tokens:**   * Use deploy SSH keys per-repository basis based on the requirement in case the access to repo is required for purposes like automation, deployment, etc. * Access tokens should be created specifically for only one purpose and must have only relevant permissions set. * Access tokens should be shared with neither anyone nor anywhere * It is always recommended to rotate access tokens periodically to mitigate risk associated with leakage.   **Handling committed keys:**   * Before importing a project into a GitHub repo, fully audit the history for sensitive data, and remove it before adding to GitHub. * If a credential was committed by mistake, do not attempt to create a new commit to hide it. Rather, rotate the credential/key and then create a new commit by removing it or delete the repository itself.   **Securing the forked repositories:**   * We use lot of forked repositories under /freshdesk GitHub account.A forked repository will be public by default. Ideally, we should create a clone into a private repository rather than a fork.     For more details please refer [Github Security Best Practices document here](https://docs.google.com/document/d/18_uxVcVzlWrlz-XXtAmg9-ossHg4RQQcCl-17jfUUyw/edit#heading=h.xd2i7e7ak7xl) |