**Bug bounty methodology**

* **Bug bounty infrastructure setup**
  + **Purchasing a server for Pentesting and Bug bounty**
    - 1 windows vps (will be used to run burp scanner continuously )
    - 2 linux vps (aws provide good vps servise , digital ocean && will be used to run all enum tools on kali such as sudomain enum , directory bruteforcing , nmap ,...)
  + **Install the tools Needed for Testing**
    - **Burp suite/ZAP proxy**
      * **configure Burp suite**
        + **identify the target scope**

target->scope->add the website you want to test

target ->sitemap->click on the long phrase -> show only in scope items // to show only the target site in the site mapper

proxy->options->intercept client request->check on is in target scope  // to only intercept target requests

* + - * + **Filter out noise in Burp Suite**

add these patterns into the Burp Suite > Proxy > Options > TLS Pass Through settings:

.\*\.google\.com

.\*\.gstatic\.com

.\*\.googleapis\.com

.\*\.pki\.goog

.\*\.mozilla\..\*

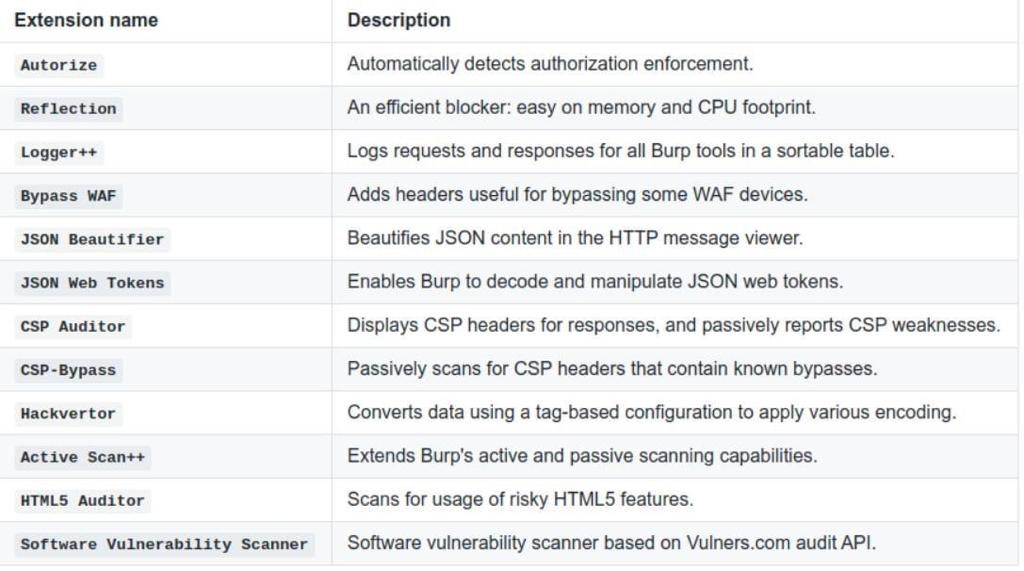
* + - * + **configure the scanner to automate and identify the vulnerabilities to scan**

scanner ->options -> active scanning area -> choose the vuln you want the scanner to search for

scanner -> live scanning ->use suite scope [defined in target scope]

// to only scan the selected target website identified in target scope

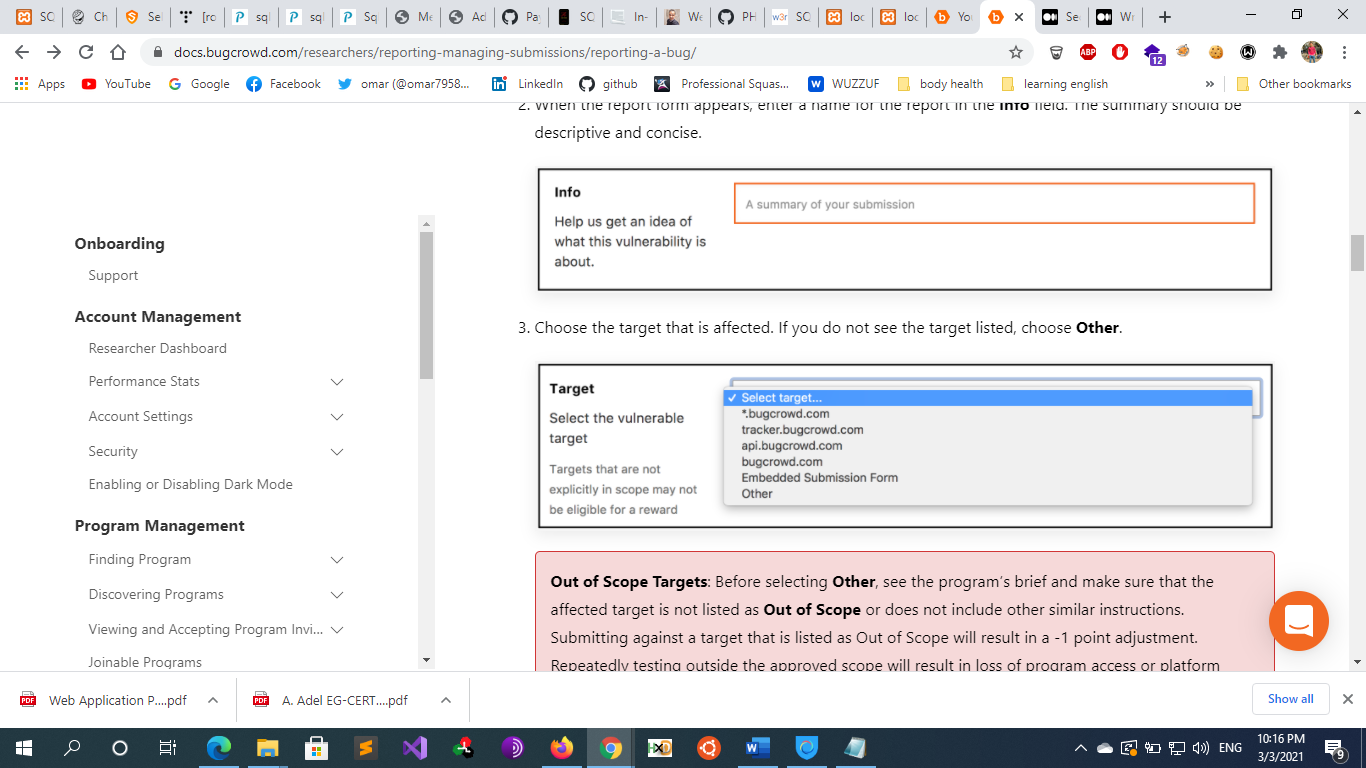
scanner is not good at finding logical vulns and access control vulnerabilities

* + - * **Burp plugins**
        + Here are the top Burp extensions our Red team uses:
        + Faraday
        + Logger ++
        + Decoder improved
        + Autorize
        + Active scan ++
        + Collaborator everywhere
        + Content-type converter
        + GraphQL RaiderHTTP
        + Request Smuggler
        + J2EEScan (good extension for java technologies)
        + Log4Shell Everywhere
        + RetireJS
        + TurboIntruder
        + Wsdler
        + upload scanner
        + paramMiner = bruteforce parameter in every page
        + Telewreck : test for weak credentials in asp
        + xss validator
        + Java Deserialization Scanner (DS)
        + Heartbleed
        + 
    - **Browser Extensions**
      * Wappalyzer
      * FoxyProxy
      * HackTools
      * EditThisCookie
      * Shodan extension
      * WayBack machine
      * hackBar
        + <https://chrome.google.com/webstore/detail/hackbar/ginpbkfigcoaokgflihfhhmglmbchinc>
        + <https://chrome.google.com/webstore/detail/hackbar/djmoeoifnlhjolebkehmpaocfnipknbh/related?hl=en-US>
        + https://chrome.google.com/webstore/detail/hackbar/gnolpldocljglopcgjfiiepkjacndoco?hl=en-US
      * User-Agent Switcher (Firefox)
    - **Dependencies**
      * Python/Go
    - **Other Tools**
      * Sublist3r
      * amass
      * Ffuf
      * Aquatone
      * Http probe
    - **All in one tools**
      * Git clone <https://github.com/nahamsec/bbht>
      * Git clone <https://github.com/SecFathy/Bugzee.git>
* **Organizing Folder structure**
  + Make a directory called "recon" and make a directory inside recon for every program you test
    - Ex : recon/hackerone
* **Mapping the application**
  + **subdomain enumeration**
  + **get IPs for subdomain founds and scan for open ports in each ip**
  + **crawling the website**
  + **directory brute forcing**:
    - to search for hidden files, backup files , and admin interfaces
    - https://pentest-tools.com/website-vulnerability-scanning/discover-hidden-directories-and-files
  + **know the technology used on the site**
  + **look at source code**
  + **Try to understand the functionality of the website**
  + **Extra work**
    - search for sensitive info in github repositories (gitgraber is a tool that automate this process)
    - python gitgraber.py -k wordlist.txt -q "companyname"
    - search in internet archive for old version of website
* **Search for vulnerabilities**
  + **First Note** 
    - There are 2 particular ways to test a web application for security vulnerabilities:
      * The first way is by going through every page and testing all the functionality. This would involve going through every page on the application, and depending on the functionality, testing for all the bugs/vulnerabilities that apply to the page. In this case, if we start off at the home page, we would try see what functionality we can exploit on the home page, and then move on to every page.
      * The second way is by breaking down the testing into different stages (including but not limited to):
        + Authorization
        + Authentication
        + Injection
        + Client-Side Controls
        + Application Logic
  + **Test for authentication mechanism vulnerabilities**
    - Test for user enumeration
    - Test for authentication bypass
    - Test for brute force protection
    - Test password quality rules
    - Test remember me functionality
    - Test for autocomplete on password forms/input
    - Test password reset and/or recovery
    - Test password change process
    - Test CAPTCHA
    - Test multi factor authentication
    - Test for logout functionality presence
    - Test for cache management on HTTP (eg Pragma, Expires, Max-age)
    - Test for default logins
    - Test for user-accessible authentication history
    - Test for out-of channel notification of account lockouts and successful password changes
    - Test for consistent authentication across applications with shared authentication schema / SSO
  + **Test for access control logical vulnerabilities manually**
    - try to access administrative pages
    - try to have multiple accounts to test access controll and check if one account can access data of other accounts
    - Test for path traversal
    - Test for bypassing authorization schema
    - Test for vertical Access control problems (a.k.a. Privilege Escalation)
    - Test for horizontal Access control problems (between two users at the same privilege level)
    - Test for missing authorization
  + **test for session management vulnerabilities**
    - token generation
      * test for token meaning
      * test for token predictability
    - token handling
      * check for insecure transmission
    - test session fixation
    - test session termination
      * Check session termination after a maximum lifetime
      * Check session termination after relative timeout
      * Check session termination after logout
    - **Check session tokens for cookie flags (httpOnly and secure)**
    - Check session cookie scope (path and domain)
    - Check session cookie duration (expires and max-age)
    - test for csrf
    - Extra
      * Establish how session management is handled in the application (eg, tokens in cookies, token in URL)
      * Test to see if users can have multiple simultaneous sessions
      * Confirm that new session tokens are issued on login, role change and logout
      * Test for consistent session management across applications with shared session management
      * Test for session puzzling
  + **Automate other vulnerabilities with scanner (input-based vulnerabilities)**
    - Test for Cross Site Scripting
    - Test for Cross Site Flashing
    - Test for HTML Injection
    - Test for SQL Injection
    - Test for LDAP Injection
    - Test for ORM Injection
    - Test for XML Injection
    - Test for XXE Injection
    - Test for SSI Injection
    - Test for XPath Injection
    - Test for XQuery Injection
    - Test for IMAP/SMTP Injection
    - Test for Code Injection
    - Test for Expression Language Injection
    - Test for Command Injection
    - Test for Overflow (Stack, Heap and Integer)
    - Test for Format String
    - Test for incubated vulnerabilities
    - Test for HTTP Splitting/Smuggling
    - Test for HTTP Verb Tampering
    - Test for Open Redirection
    - Test for Local File Inclusion
    - Test for Remote File Inclusion
    - Compare client-side and server-side validation rules
    - Test for NoSQL injection
    - Test for HTTP parameter pollution
    - Test for auto-binding
    - Test for Mass Assignment
    - Test for NULL/Invalid Session Cookie
  + **Test Other Bugs**
    - **Risky Functionality - File Uploads**
      * Test that acceptable file types are whitelisted
      * Test that file size limits, upload frequency and total file counts are defined and are enforced
      * Test that file contents match the defined file type
      * Test that all file uploads have Anti-Virus scanning in-place.
      * Test that unsafe filename are sanitized
      * Test that uploaded files are not directly accessible within the web root
      * Test that uploaded files are not served on the same hostname/port
      * Test that files and other media are integrated with the authentication and authorization schemas
    - **Configuration Management**
      * Check for commonly used application and administrative URLs
      * Check for old, backup and unreferenced files
      * Check HTTP methods supported and Cross Site Tracing (XST) [Allow **Trace** Method]
      * Test file extensions handling
      * **Test for security HTTP headers (e.g. CSP, X-Frame-Options, HSTS)**
      * Test for policies (e.g. Flash, Silverlight, robots)
      * Test for non-production data in live environment, and vice-versa
      * Check for sensitive data in client-side code (e.g. API keys, credentials)
    - **Denial of Service**
      * Test for anti-automation
      * Test for account lockout
      * Test for HTTP protocol DoS
      * Test for SQL wildcard DoS
    - **Business Logic**
      * Test for feature misuse
      * Test for lack of non-repudiation
      * Test for trust relationships
      * Test for integrity of data
      * Test segregation of duties
    - **HTML 5**
      * Test Web Messaging
      * Test for Web Storage SQL injection
      * Check CORS implementation
      * Check Offline Web Application
    - **Risky Functionality - Card Payment**
      * Test for known vulnerabilities and configuration issues on Web Server and Web Application
      * Test for default or guessable password
      * Test for non-production data in live environment, and vice-versa
      * Test for Injection vulnerabilities
      * Test for Buffer Overflows
      * Test for Insecure Cryptographic Storage
      * Test for Insufficient Transport Layer Protection
      * Test for Improper Error Handling
      * Test for all vulnerabilities with a CVSS v2 score > 4.0
      * Test for Authentication and Authorization issues
      * Test for CSRF
    - **Secure Transmission**
      * Check SSL Version, Algorithms, Key length
      * Check for Digital Certificate Validity (Duration, Signature and CN)
      * **Check credentials only delivered over HTTPS**
      * **Check that the login form is delivered over HTTPS**
      * **Check session tokens only delivered over HTTPS**
      * **Check if HTTP Strict Transport Security (HSTS) in use**
    - **Cryptography**
      * Check if data which should be encrypted is not
      * Check for wrong algorithms usage depending on context
      * Check for weak algorithms usage
      * Check for proper use of salting
      * Check for randomness functions
  + **Low risk bugs**
    - **WEEK password POLICY in Register page**
      * **Summary**
      * **Remediation**
    - **MISSING RATE LIMIT in Login page**
      * **Summary**
      * **Remediation**
      * Allow for brute-forcing
    - **User/email enumeration** 
      * **Summary**
      * **Remediation**
    - **MISSING IMPORTANT SECURITY FLAGS** 
      * **Summary**
      * **Remediation**
      * Missing X-Frame-Options
    - **COOKIE WITHOUT HTTPONLY FLAG SET**
      * **Summary**
      * **Remediation**
    - **COOKIE WITHOUT secure FLAG SET**
      * **Summary**
      * **Remediation**
    - **Password Autocomplete**
      * **Summary**
        + An attacker with local access to the PC or by compromising the browser could obtain the clear text password from the browser saved password.
      * **Remediation** 
        + To prevent browsers from storing credentials entered into HTML forms, include the attribute autocomplete="off" within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields)
    - **Check for secure Transmission**
      * **Check credentials only delivered over HTTPS**
      * **Check that the login/registration form is delivered over HTTPS**
      * **Check session tokens only delivered over HTTPS**
      * **Missing HTTP Strict Transport Security (HSTS)** 
        + **summary**

It has been discovered that the affected application is using HTTPS, however does not use the HSTS header. The HTTP protocol by itself is clear text, meaning that any data that is transmitted via HTTP can be captured and the contents viewed. To keep data private and prevent it from being intercepted, HTTP is often tunneled through either Secure Sockets Layer (SSL) or Transport Layer Security (TLS). When either of these encryption standards are used, it is referred to as HTTPS.

HTTP Strict Transport Security (HSTS) is an optional response header that can be configured on the server to instruct the browser to only communicate via HTTPS. This will be enforced by the browser even if the user requests a HTTP resource on the same server.

Cyber-criminals will often attempt to compromise sensitive information passed from the client to the server using HTTP. This can be conducted via various Man-in-The-Middle (MiTM) attacks or through network packet captures.

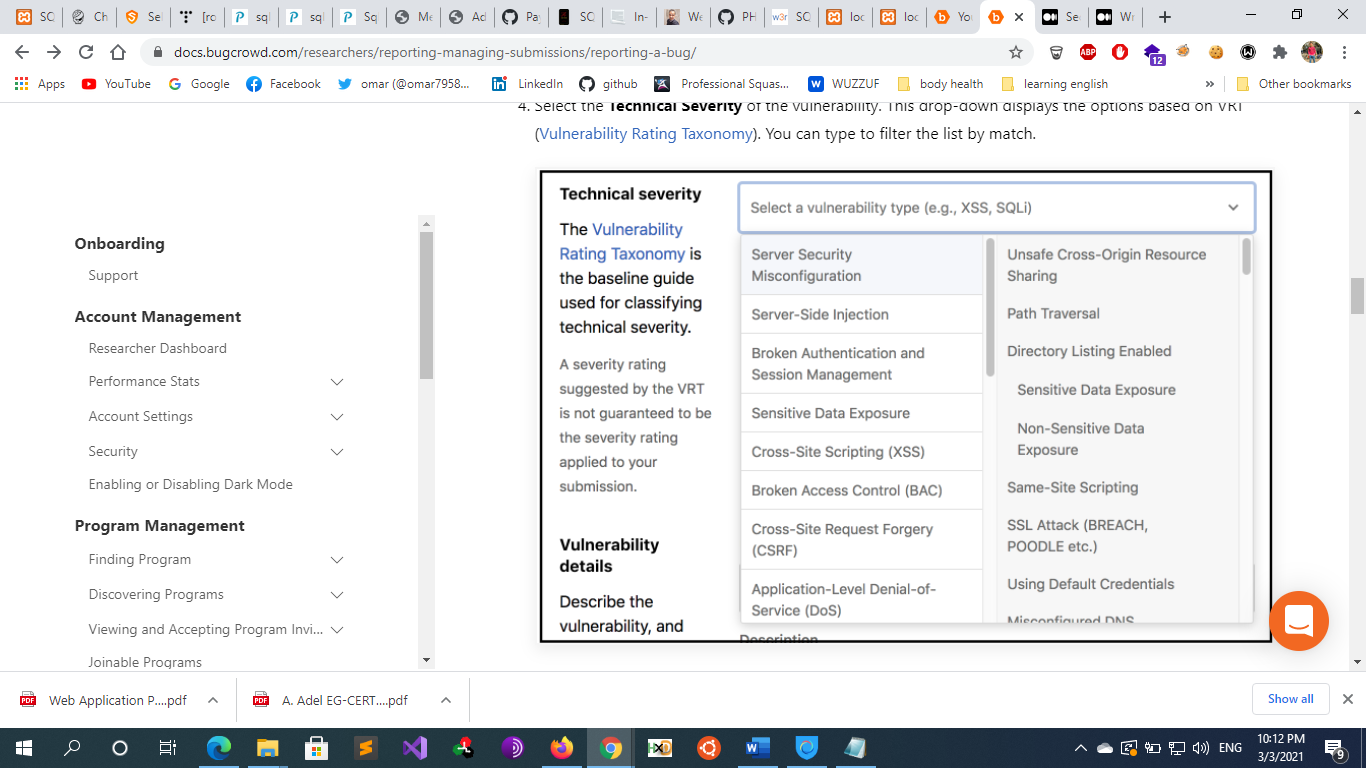
* + - **click-jacking**
      * **summary**
        + If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.
      * **Remediation**
        + Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker
    - **SESSION FIXATION**
      * **Remediation**
* **Exploiting the vulnerabilities found**
  + just poc (proof of concept) don't harm the web app
* **Report the vulnerabilities**
  + There is not a prescribed or standard structure to penetration testing reports. However, there are best practices, do’s and don’ts and critical aspects that should be taken care of while writing a report.
  + Generally, you have to explain where the bug was found, who it affects, how to reproduce it, the parameters it affects, and provide Proof-of-Concept supporting information. You can upload any files or logs as supporting evidence. This not only helps quickly reproduce the issue but moves your submission through the review process faster, with no delays due to missing information.
  + **The report must contain the following information at a minimum:**
    - **Title**
      * When creating a title for the vulnerability, be explicit about what the vulnerability is. Don’t write show-off titles. Let’s consider an cross-site scripting example (XSS vulnerability example):
        + Example of a good title — Reflected XSS on reference parameter at product page
        + Example of a bad title — CRITICAL — XSS on your program
      * Keep in mind that this is the first thing the program owners or clients will see. It’s their first impression of you and your report.
    - **Target** 
      * The Target field identifies the specific target affected by the bug you have found.
      * 
    - **Technical Severity**

## ****Impact of the vulnerability****

## The impact of the vulnerability reflects the severity of the report. By referring to the result of the attack, try to explain what the attacker can do, what information they can access and how this affects other users in the system. The higher the severity, the higher the awarded bounty.

* + - * **Scoring System** 
        + There are multiple Vulnerabilities priority rating scoring system such as CVSS and VRT
        + Bug crowd’s Vulnerability Rating Taxonomy (VRT) is a resource outlining Bugcrowd’s baseline priority rating, The Vulnerability Rating Taxonomy Classification identifies the kind of bug you have found based on our VRT, our baseline priority rating system for common bugs found on bug bounty programs. It is important that you choose the correct type so that the organization understands the risk the bug presents them.
        + vulnerability rating system created by Bugcrowd called the Vulnerability Rating Taxonomy (VRT). is an attempt to systematically assess a vulnerability's severity . The VRT is also compatible with another attempt at providing a common threat metric, the Common Vulnerability Scoring System (CVSS) . VRT can be used to calculate CVSS. Understanding the VRT can help you direct your efforts to bugs that will give you the most value for your time.

https://bugcrowd.com/vulnerability-rating-taxonomy

* + - * + 
    - **Vulnerability details**
      * This section should include the following information:
        + **Bug URL**: The bug URL identifies the location in the application where you discovered the bug.
        + **Description:**

A vulnerability description must be short, clear, and direct. Program owners and clients don’t want to spend much time reading.

A great way to describe a vulnerability in a short, clear way is to include references/links to trusted sources that can help others understand, identify, and fix the bug. This could be an OWASP link, CVE references or links to other public advisories and standards. (Don’t reference Wikipedia or other less respected sites.)

e.g. If I find a XSS bug, I’ll explain what it is, give an OWASP reference, tell them what type of XSS was found, and so on.

Don’t copy-paste information from automated tools or other sources into the description. It shows the program owners or clients that you didn’t even take the time to write a few words specific to their scenario

* + - * + **Proof Of Concept:**

**steps of Reproduction**:

provide a clear step-by-step guide or process showing how to replicate the vulnerability.

Example of a XSS proof of concept:

Step 1: Go to the following [URL]

Step 2: Enter your username and password (you need an account to do this)

Step 3: In the Search box at top right, insert the following information:

<script>alert(document.domain);</script>

Step 4: Click the “Search” button

Step 5: You’ll see a JavaScript popup box showing your domain

Check the attached screenshot to see the actual XSS vulnerability.

Depending on the type of vulnerability, I may also show the reflected code so the program owners and clients can identify faster where the payload is loading..

**Screenshots or videos**: Provide illustrative evidence in the form of screenshots or videos that shows proof of the vulnerability. This is one of the most impactful things you can do to provide context around your submission. We strongly recommend you provide this every time you submit.

try to take notes with everything you have found and screen shots as many as you can

## ****Suggested Fixes/Solutions****

* + - * Show the program owners clear solutions for their problem. Give examples, don’t just tell them to sanitize the input, but also give them references and possible ways to do it. They’ll greatly appreciate it. Sometimes the developers don’t know how to fix a vulnerability, and if you provide a great description of a suggested fix it’s a win-win situation. After all, the key mission is to fix the vulnerabilities.
  + **Critical Information: What Your Report Needs**
    - **Although report information will vary based on what the vulnerability, there is a common set of fields you will always need:**
      * The location (URL) of the vulnerability
      * The vulnerability types
      * When it was found
      * How it was found (automated/manual, tool)
      * How to reproduce it
      * How the bug can be exploited
      * How to mitigate (if possible)
  + **Differences Between Bug Bounty Reports and Pentest Reports**
    - A bug bounty report documents a single vulnerability while a pentest report documents all discovered vulnerabilities. For instance, an average application pentest discovers 20 to 30 vulnerabilities.
    - Pentest Executive Summary — Pentest reports typically include an Executive Summary near the beginning to provide a testing overview and the security tester’s impression of overall security risk. The Executive Summary also notes any trends in the types of weaknesses found; for instance, if several weaknesses fall under an OWASP Top 10 category, it would be noted. The Summary also could include information from the project’s Statement of Work such as assessment scope and methodologies. Ideally, the Summary is written in less technical terms to encourage distribution beyond the IT and security teams to business and management stakeholders. After all, security risk is ultimately business risk. The Summary also typically includes a list or table of discovered vulns
  + **Reports Templates**

## <https://github.com/DFC302/BugBountyTemplate?fbclid=IwAR29QBjsayymOt6IgOJSh0iggRLWvwupt5o5_HpeByR1Ngjn60hQELuxXXo>

* + - <https://github.com/gwen001/bb-reports-templates>
    - https://github.com/ZephrFish/BugBountyTemplates
    - https://github.com/fransr/template-generator
    - https://buer.haus/breport/
  + **Resources**
    - <https://docs.bugcrowd.com/researchers/reporting-managing-submissions/reporting-a-bug/>
    - <https://bugbountyguide.com/hunters/proof-of-concepts.html?fbclid=IwAR13q17GtdQiP87yW0O5nZwrfT_rkhZYXn_tEHnBoyxc8Tl-WlLlADNyHNA>
    - <http://www.pentest-standard.org/index.php/Reporting>
    - <https://cure53.de/#publications>
    - <https://medium.com/@tolo7010/writing-a-good-and-detailed-vulnerability-report-bdb86cedcff>
    - <https://blog.cobalt.io/how-to-write-a-great-vulnerability-report-ab8654c6290c>
* **Notes**
  + **bug bounty programs:**
    - **Third Party Marketplace :** Marketplaces are sites that match companies and researchers
      * hackerone
      * bugcrowd
      * synack
      * cobalt
      * zerocopter
      * vulnerability lab
    - **Company-sponsored programs**
      * Google
      * Facebook
      * Github
      * Amazon
      * Microsoft
  + **whois info** :
    - when someone buy a domain name a data base stores all registration details that includes :
      * name of the organization that the domain belongs to
      * organization owner purchasing the domain
      * name of the developer who handles the website
      * and other info like address , emails , contact numbers , full names
      * these info cant be fully proofed as the registrant can fill wrong information
  + **Tips and tricks**
    - Find hidden GET parameters in javascript files
      * assetfinder example.com | gau | egrep -v '(.css|.png|.jpeg|.jpg|.svg|.gif|.wolf)' | while read url; do vars=$(curl -s $url | grep -Eo "var [a-zA-Z0-9]+" | sed -e 's,'var','"$url"?',g' -e 's/ //g' | grep -v '.js' | sed 's/.\*/&=xss/g'); echo -e "\e[1;33m$url\n\e[1;32m$vars"; done
    - There is a new tool in town called bcscope which can get you the scope of all bug bounty programs available on Bugcrowd platform, including the private ones.All you have to do is to provide your Bugcrowd token like this:
      * bcscope -t <YOUR-TOKEN-HERE> -c 2 -p
    - **Top 20 search engines for hackers**
      * censys.io – Censys Search Engine
      * shodan.io – Search engine for Internet-connected devices
      * viz.greynoise.io – GreyNoise Visualizer
      * zoomeye.org – Cyberspace Search Engine
      * onyphe.io – Cyber Defense Search Engine
      * wigle.net – Wireless Network Mapping
      * intelx.io – Intelligence X
      * fofa.so – Cyberspace Security Search Engine
      * hunter.io – OSINT Search Engine
      * zorexeye.com – Hacker’s Search Engine
      * pulsedive.com – Threat intelligence Search Engine
      * netograph.io – Mapping the deep structure of the web
      * vigilante.pw – Breached Database Directory
      * pipl.com – Search engine for real identity profiles
      * abuse.ch – Threat intelligence, malwares etc.
      * maltiverse.com – Open IoC Search Engine, Threat Intel
      * insecam.org – World biggest online cameras directory
      * spyse.com – Internet Assets Search Engine
      * dnsdumpster.com – DNS recon & research
      * phonebook.cz – Search for subdomains, email addresses, or URLs
  + Greenshot :
    - is a good tool to take sc for poc
  + autohotkey :
    - automate some payloads or keyboards keys
  + lazyReconn is a tool written by naham sec that combine many reconn tools in one tool
  + s3scanner :
    - tool for s3 buckets takeover (insecure aws buckets)
  + run burp in the background with the following extentions {j2EEscan ,Telewreck , paramMiner  , uploadScanner ,  Nginx alias Traversal}
  + subdomain takeover tools
    - <https://github.com/haccer/subjack>
    - <https://github.com/m4ll0k/takeover>
  + [xsshunter.com](http://xsshunter.com/) : for out of band xss (user-agent and/or x-forwarded-for)
    - configure xsshunter with burp
      * proxy->options->match and replace (burp can change some headers evert time it send a requests by this option)
        + replace your xss payload in the user agent
  + easiest way to put your name on the wall of fame of google is to test on google acquisitions (companies google bought)

+] Information Gathering

Manually explore the site

Spider/crawl for missed or hidden content

Check for files that expose content, such as robots.txt, sitemap.xml, .DS\_Store

Check the caches of major search engines for publicly accessible sites

Check for differences in content based on User Agent (eg, Mobile sites, access as a Search engine Crawler)

Perform Web Application Fingerprinting

Identify technologies used

Identify user roles

Identify application entry points

Identify client-side code

Identify multiple versions/channels (e.g. web, mobile web, mobile app, web services)

Identify co-hosted and related applications

Identify all hostnames and ports

Identify third-party hosted content