Bug Bounty Hunting Process

Bug Bounty Programs

As mentioned in this module's summary, we usually consider a bug bounty program as a crowdsourcing initiative through which individuals can receive recognition and compensation for discovering and reporting software bugs.

Bug bounty programs are more than that, though. A bug bounty program (also called a vulnerability rewards program - VRP) is continuous and proactive security testing that supplements internal code audits and penetration tests and completes an organization's vulnerability management strategy.

HackerOne aptly describes their bug bounty platform (that can host bug bounty programs) as "Continuous testing, constant protection" and as something that can be integrated seamlessly into an organization's existing development life cycle.

Bug Bounty Program Types

A bug bounty program can be `private` or `public`.

- `Private bug bounty programs` are not publicly available. Bug bounty hunters can only participate in a private bug bounty program upon receiving specific invitations. The vast majority of bug bounty programs start as private ones and become public after getting the hang of receiving and triaging vulnerability reports.
    - Most of the time, bug bounty hunters receive invitations to private bug bounty programs based on their track record, valid finding consistency, and violation record. A representative example of this is how `HackerOne` deals with invitations based on specific criteria. Please note that certain bug bounty programs may even require a background check.
- `Public bug bounty programs` are accessible by the entire hacking community.
- Parent/Child Programs also exist where a bounty pool and a single cyber security team are shared between a parent company and its subsidiaries. If a subsidiary launches a bug bounty program (child program), this program will be linked to the parent one.

Something important to note is that the terms `Bug Bounty Program (BBP)` and `Vulnerability Disclosure Program (VDP)` should not be used interchangeably.

A vulnerability disclosure program only provides guidance on how an organization prefers receiving information on identified vulnerabilities by third parties. A bug bounty program incentivizes third parties to discover and report software bugs, and bug bounty hunters receive monetary rewards in return.

If you want to study the anatomy of a vulnerability disclosure program, refer to the following resource. VDP vs. BBP

Bug Bounty Program Code of Conduct

The violation record of a bug bounty hunter is always taken into consideration. For this reason, it is of paramount importance to adhere to the code of conduct/policy of each bug bounty program or bug bounty platform. Spend considerable time reading the code of conduct as it does not just establish expectations for behavior but also makes bug bounty hunters more effective and successful during their bug report submissions.

If you want to become an established bug bounty hunter, you will have to strike a balance between professionalism and technical capability.

We strongly suggest that you go over HackerOne's Code of Conduct to familiarize yourself with such documents.

Bug Bounty Program Structure

It is about time we see what a bug bounty program looks like. Navigate to HackerOne's bug bounty program list to go over some bug bounty programs. Take Alibaba BBP and Amazon Vulnerability Research Program as examples and go through their "Policy."

According to HackerOne: The policy section enables organizations to publish information about their program to communicate the specifics about their program to hackers. Organizations typically publish a vulnerability disclosure policy with guidance on how they want to receive information related to potential vulnerabilities in their products or online services.
The policy also includes the program's scope, which lists items hackers can test and send reports in for. It is often defined by the domain name for web applications or by the specific App Store / Play store mobile apps that a company builds.

A bug bounty program usually consists of the following elements:

`Vendor Response SLAs`	Defines when and how the vendor will reply
`Access`	Defines how to create or obtain accounts for research purposes
`Eligibility Criteria`	For example, be the first reporter of a vulnerability to be eligible, etc.
`Responsible Disclosure Policy`	Defines disclosure timelines, coordination actions to safely disclose a vulnerability, increase user safety, etc.
`Rules of Engagement`	
`Scope`	In-scope IP Ranges, domains, vulnerabilities, etc.
`Out of Scope`	Out-of-scope IP Ranges, domains, vulnerabilities, etc.
`Reporting Format`	
`Rewards`	
`Safe Harbor`	
`Legal Terms and Conditions`	
`Contact Information`	

In HackerOne's case, the above are usually included inside the `Policy` part of each program.

Please go over a bug bounty program's description/policy meticulously. The same goes for any "code of conduct" documents they may include. By doing so, you can meet expectations and avoid unnecessary back and forth that could cause significant time loss. In bug bounty hunting, time is of the essence!

Finding Bug Bounty Programs

One of the best online resources to identify bug bounty programs of your liking is HackerOne's Directory. HackerOne's directory can be used for identifying both organizations that have a bug bounty program and contact information to report vulnerabilities you have ethically found.

Enable step-by-step solutions for all questions

“8ks0kL9iBUvF” is not created yet. Click to create.

Questions

Answer the question(s) below
to complete this Section and earn cubes!

+ 5 Which Bug Bounty Program part establishes expectations for behavior while participating in a bug bounty program?

Submit : `Code of conduct`

Hint

Writing a Good Report

By documenting our findings clearly and concisely, we get straight to our point in a way that the security or triage team can comprehend. Most importantly, bug reports should include information on how exploitation of each vulnerability can be reproduced step-by-step.

Please note that when reporting to less mature companies, we may have to translate technical security issues into more understandable/business terms for them to understand the actual impact of each vulnerability.

The essential elements of a good bug report are (the element order can vary):

`Vulnerability Title`	Including vulnerability type, affected domain/parameter/endpoint, impact etc.
`CWE & CVSS score`	For communicating the characteristics and severity of the vulnerability.
`Vulnerability Description`	Better understanding of the vulnerability cause.
`Proof of Concept (POC)`	Steps to reproduce exploiting the identified vulnerability clearly and concisely.
`Impact`	Elaborate more on what an attacker can achieve by fully exploiting the vulnerability. Business impact and maximum damage should be included in the impact statement.
`Remediation`	Optional in bug bounty programs, but good to have.

Readable and well-formatted bug reports can drastically minimize both vulnerability reproduction time and time to triage.

Why CWE & CVSS?

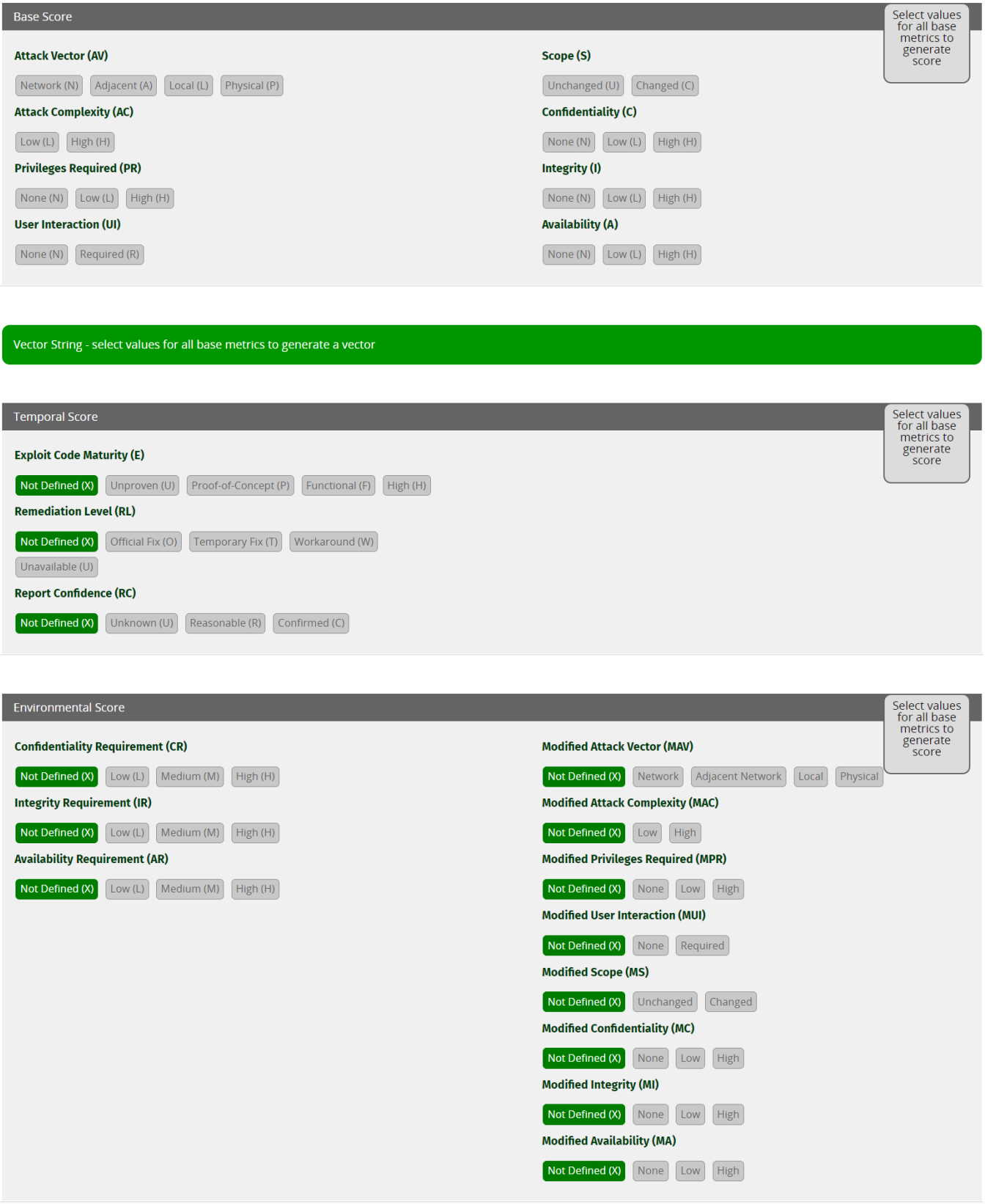
MITRE describes Common Weaknesses Enumeration (CWE) as a community-developed list of software and hardware weakness types. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts. In the case of a vulnerability chain, choose a CWE related to the initial vulnerability.

When it comes to communicating the severity of an identified vulnerability, then Common Vulnerability Scoring System (CVSS) should be used, as it is a published standard used by organizations worldwide.

Using CVSS Calculator

Let us now see how we can use the CVSS v3.1 Calculator identify the severity of an identified vulnerability.

We will focus on the `Base Score` area only.



Attack Vector

Shows how the vulnerability can be exploited.

- `Network (N):` Attackers can only exploit this vulnerability through the network layer (remotely exploitable).
- `Adjacent (A):` Attackers can exploit this vulnerability only if they reside in the same physical or logical network (secure VPN included).
- `Local (L):` Attackers can exploit this vulnerability only by accessing the target system locally (e.g., keyboard, terminal, etc.) or remotely (e.g., SSH) or through user interaction.
- `Physical (P):` Attackers can exploit this vulnerability through physical interaction/manipulation.

Attack Complexity

Depicts the conditions beyond the attackers' control and must be present to exploit the vulnerability successfully.

- `Low (L):` No special preparations should take place to exploit the vulnerability successfully. The attackers can exploit the vulnerability repeatedly without any issue.
- `High (H):` Special preparations and information gathering should take place to exploit the vulnerability successfully.

Privileges Required

Show the level of privileges the attacker must have to exploit the vulnerability successfully.

- `None (N):` No special access related to settings or files is required to exploit the vulnerability successfully. The vulnerability can be exploited from an unauthorized perspective.
- `Low (L):` Attackers should possess standard user privileges to exploit the vulnerability successfully. The exploitation in this case usually affects files and settings owned by a user or non-sensitive assets.
- `High (H):` Attackers should possess admin-level privileges to exploit the vulnerability successfully. The exploitation in this case usually affects the entire vulnerable system.

User Interaction

Shows if attackers can successfully exploit the vulnerability on their own or user interaction is required.

- `None (N):` Attackers can successfully exploit the vulnerability independently.
- `Required (R):` A user should take some action before the attackers can successfully exploit the vulnerability.

Scope

Shows if successful exploitation of the vulnerability can affect components other than the affected one.

- `Unchanged (U):` Successful exploitation of the vulnerability affects the vulnerable component or affects resources managed by the same security authority.
- `Changed (C):` Successful exploitation of the vulnerability can affect components other than the affected one or resources beyond the scope of the affected component's security authority.

Confidentiality

Shows how much the vulnerable component's confidentiality is affected upon successfully exploiting the vulnerability. Confidentiality limits information access and disclosure to authorized users only and prevents unauthorized users from accessing information.

- `None (N):` The confidentiality of the vulnerable component does not get impacted.
- `Low (L):` The vulnerable component will experience some loss of confidentiality upon successful exploitation of the vulnerability. In this case, the attackers do not have control over what information is obtained.
- `High (H):` The vulnerable component will experience total (or serious) loss of confidentiality upon successfully exploiting the vulnerability. In this case, the attackers have total (or some) control over what information is obtained.

Integrity

Shows how much the vulnerable component's integrity is affected upon successfully exploiting the vulnerability. Integrity refers to the trustworthiness and veracity of information.

- `None (N):` The integrity of the vulnerable component does not get impacted.
- `Low (L):` Attackers can modify data in a limited manner on the vulnerable component upon successfully exploiting the vulnerability. Attackers do not have control over the consequence of a modification, and the vulnerable component does not get seriously affected in this case.
- `High (H):` Attackers can modify all or critical data on the vulnerable component upon successfully exploiting the vulnerability. Attackers have control over the consequence of a modification, and the vulnerable component will experience a total loss of integrity.

Availability

Shows how much the vulnerable component's availability is affected upon successfully exploiting the vulnerability. Availability refers to the accessibility of information resources in terms of network bandwidth, disk space, processor cycles, etc.

- `None (N):` The availability of the vulnerable component does not get impacted.
- `Low (L):` The vulnerable component will experience some loss of availability upon successfully exploiting the vulnerability. The attacker does not have complete control over the vulnerable component's availability and cannot deny the service to users, and performance is just reduced.
- `High (H):` The vulnerable component will experience total (or severe) availability loss upon successfully exploiting the vulnerability. The attacker has complete (or significant) control over the vulnerable component's availability and can deny the service to users. Performance is significantly reduced.

Examples

Find below some examples of using CVSS 3.1 to communicate the severity of vulnerabilities.

`Title:`	Cisco ASA Software IKEv1 and IKEv2 Buffer Overflow Vulnerability (CVE-2016-1287)
`CVSS 3.1 Score:`	9.8 (Critical)
`Attack Vector:`	Network - The Cisco ASA device was exposed to the internet since it was used to facilitate connections to the internal network through VPN.
`Attack Complexity:`	Low - All the attacker has to do is execute the available exploit against the device
`Privileges Required:`	None - The attack could be executed from an unauthenticated/unauthorized perspective
`User Interaction:`	None - No user interaction is required
`Scope:`	Unchanged - Although you can use the exploited device as a pivot, you cannot affect other components by exploiting the buffer overflow vulnerability.
`Confidentiality:`	High - Successful exploitation of the vulnerability results in unrestricted access in the form of a reverse shell. Attackers have total control over what information is obtained.
`Integrity:`	High - Successful exploitation of the vulnerability results in unrestricted access in the form of a reverse shell. Attackers can modify all or critical data on the vulnerable component.
`Availability:`	High - Successful exploitation of the vulnerability results in unrestricted access in the form of a reverse shell. Attackers can deny the service to users by powering the device off

`Title:`	Stored XSS in an admin panel (Malicious Admin -> Admin)
`CVSS 3.1 Score:`	5.5 (Medium)
`Attack Vector:`	Network - The attack can be mounted over the internet.
`Attack Complexity:`	Low - All the attacker (malicious admin) has to do is specify the XSS payload that is eventually stored in the database.
`Privileges Required:`	High - Only someone with admin-level privileges can access the admin panel.
`User Interaction:`	None - Other admins will be affected simply by browsing a specific (but regularly visited) page within the admin panel.
`Scope:`	Changed - Since the vulnerable component is the webserver and the impacted component is the browser
`Confidentiality:`	Low - Access to DOM was possible
`Integrity:`	Low - Through XSS, we can slightly affect the integrity of an application
`Availability:`	None - We cannot deny the service through XSS

Good Report Examples

Find below some good report examples selected by HackerOne:

- SSRF in Exchange leads to ROOT access in all instances
- Remote Code Execution in Slack desktop apps + bonus
- Full name of other accounts exposed through NR API Explorer (another workaround of #476958)
- A staff member with no permissions can edit Store Customer Email
- XSS while logging in using Google
- Cross-site Scripting (XSS) on HackerOne careers page

Please refer to the Submitting Reports section of HackerOne's docs portal for the actual process a bug bounty hunter has to follow to submit a bug report.

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below
to complete this Section and earn cubes!

+ 5 Which base metric value of the base score considers that attackers can only exploit a vulnerability if they reside in the same physical or logical network as the target host/application?

Submit : `Adjacent`

Interacting with Organizations/BBP Hosts

Suppose that you have submitted a bug report. How should you interact with the security/triage team after that?

Well, to begin with, do not interact with them. Allow the security/triage team some time to process your report, validate your finding, and maybe ask questions. Some bug bounty programs/platforms include vendor response SLAs or response efficiency metrics, which can give you an idea of how long it can take for them to get back to a submission. Also, make sure that you do not spam the security/triage team within a short period of time.

If the security/triage team does not get back to you in a reasonable amount of time, then if the submission was through a bug bounty platform, you can contact Mediation.

Once the security/triage team gets back to you, note the team member's username and tag them in any future communications since they will probably be dealing with your submission. Do not interact with the security/triage team through any unofficial communication channel (social media etc.)!

A professional bug report should be accompanied by professional communication. Remain calm and interact with the security/triage team as a security professional would.

During your interaction with the security/triage team, there could be disagreements about the severity of the bug or the bounty. A bug's impact and severity play a significant role during the bounty amount assignment. In the case of such a disagreement, proceed as follows.

- Explain your rationale for choosing this severity score and guide the security/triage team through each metric value you specified in the CVSS calculator. Eventually, you will come to an agreement.
- Go over the bug bounty program's policy and scope and ensure that your submission complies with both. Also, make sure that the bounty amount resembles the policy of the

bug bounty program.
- If none of the above was fruitful, contact mediation or a similar platform service.

Example 1: Reporting Stored XSS

`Title`: Stored Cross-Site Scripting (XSS) in X Admin Panel

`CWE`: CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

`CVSS 3.1 Score`: 5.5 (Medium)

`Description`: During our testing activities, we identified that the "X for administrators" web application is vulnerable to stored cross-site scripting (XSS) attacks due to inadequate sanitization of user-supplied data.
Specifically, the file uploading mechanism at "Admin Info" -> "Secure Data Transfer" -> "Load of Data" utilizes a value obtained from user input, specifically the uploaded file's filename, which is not only directly reflected back to the user’s browser but is also stored into the web application’s database. However, this value does not appear to be adequately sanitized. It, therefore, results in the application being vulnerable to reflected and stored cross-site scripting (XSS) attacks since JavaScript code can be entered in the filename field.

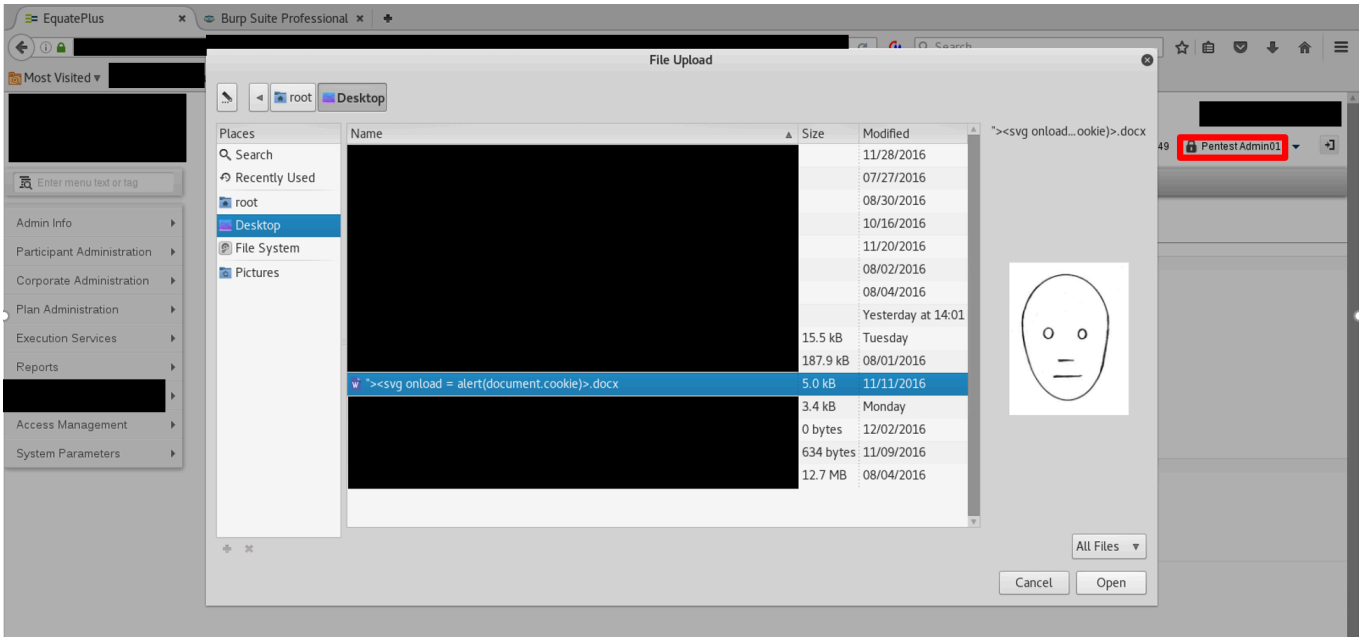
`Impact`: Cross-Site Scripting issues occur when an application uses untrusted data supplied by offensive users in a web browser without sufficient prior validation or escaping. A potential attacker can embed untrusted code within a client-side script to be executed by the browser while interpreting the page.
Attackers utilize XSS vulnerabilities to execute scripts in a legitimate user's browser leading to user credentials theft, session hijacking, website defacement, or redirection to malicious sites. Anyone that can send data to the system, including administrators, are possible candidates for performing XSS attacks against the vulnerable application.
This issue introduces a significant risk since the vulnerability resides in the "X for administrators” web application, and the uploaded files are visible and accessible by every administrator. Consequently, any administrator can be a possible target of a Cross-Site Scripting attack.

`POC`:

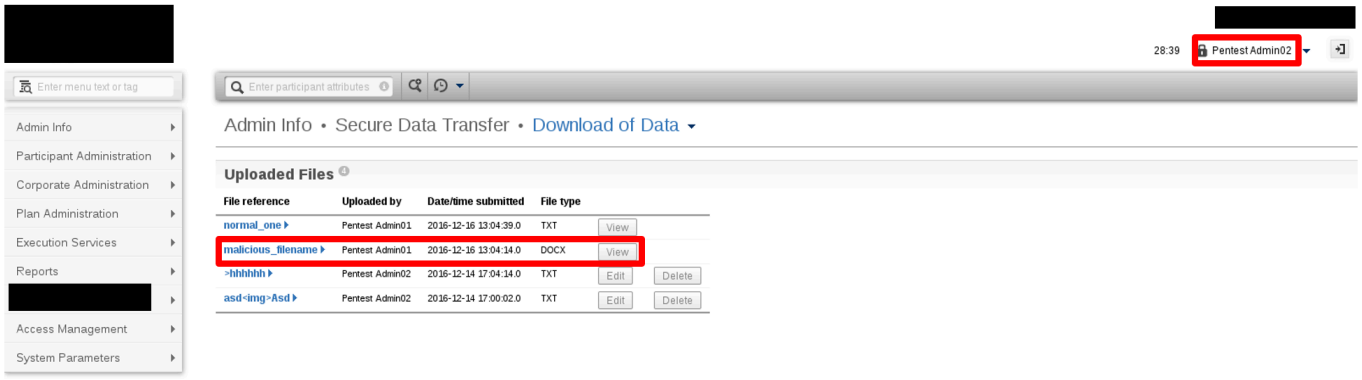
Step 1: A malicious administrator could leverage the fact that the filename value is reflected back to the browser and stored in the web application’s database to perform cross-site scripting attacks against other administrators by uploading a file containing malicious JavaScript code

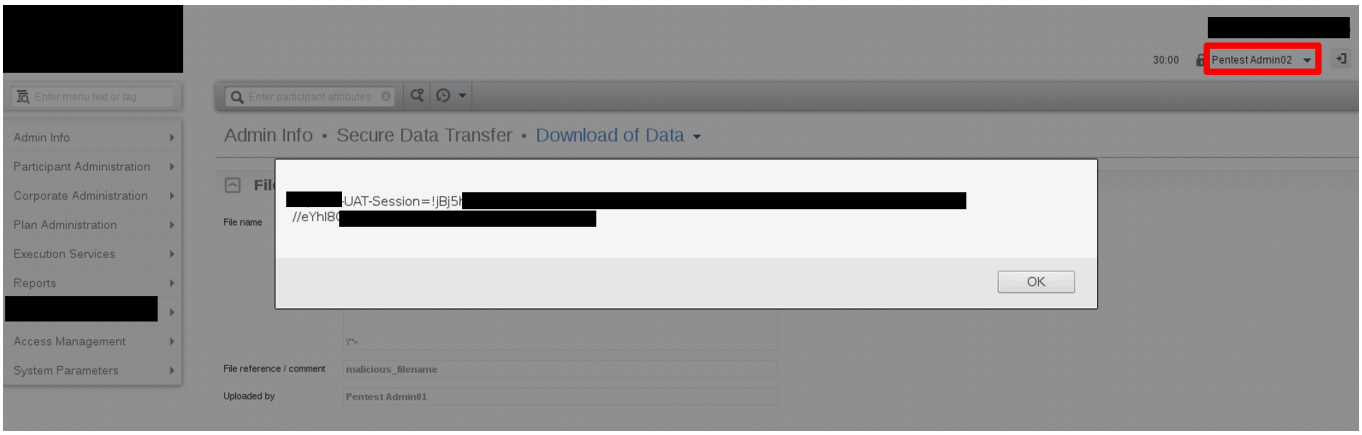
into its filename. The attack is feasible because administrators can view all uploaded files regardless of the uploader. Specifically, we named the file, as follows, using a Linux machine.

```
"><svg onload = alert(document.cookie)>.docx
```



Step 2: When another administrator clicks the view button to open the abovementioned file, the malicious JavaScript code in the file's filename will be executed on the browser.





CVSS Score Breakdown

`Attack Vector:`	Network - The attack can be mounted over the internet.
`Attack Complexity:`	Low - All the attacker (malicious admin) has to do is specify the XSS payload eventually stored in the database.
`Privileges Required:`	High - Only someone with admin-level privileges can access the admin panel.
`User Interaction:`	None - Other admins will be affected simply by browsing a specific (but regularly visited) page within the admin panel.
`Scope:`	Changed - Since the vulnerable component is the webserver and the impacted component is the browser
`Confidentiality:`	Low - Access to DOM was possible
`Integrity:`	Low - Through XSS, we can slightly affect the integrity of an application
`Availability:`	None - We cannot deny the service through XSS

Example 2: Reporting CSRF

`Title` : Cross-Site Request Forgery (CSRF) in Consumer Registration

`CWE` : CWE-352: Cross-Site Request Forgery (CSRF)

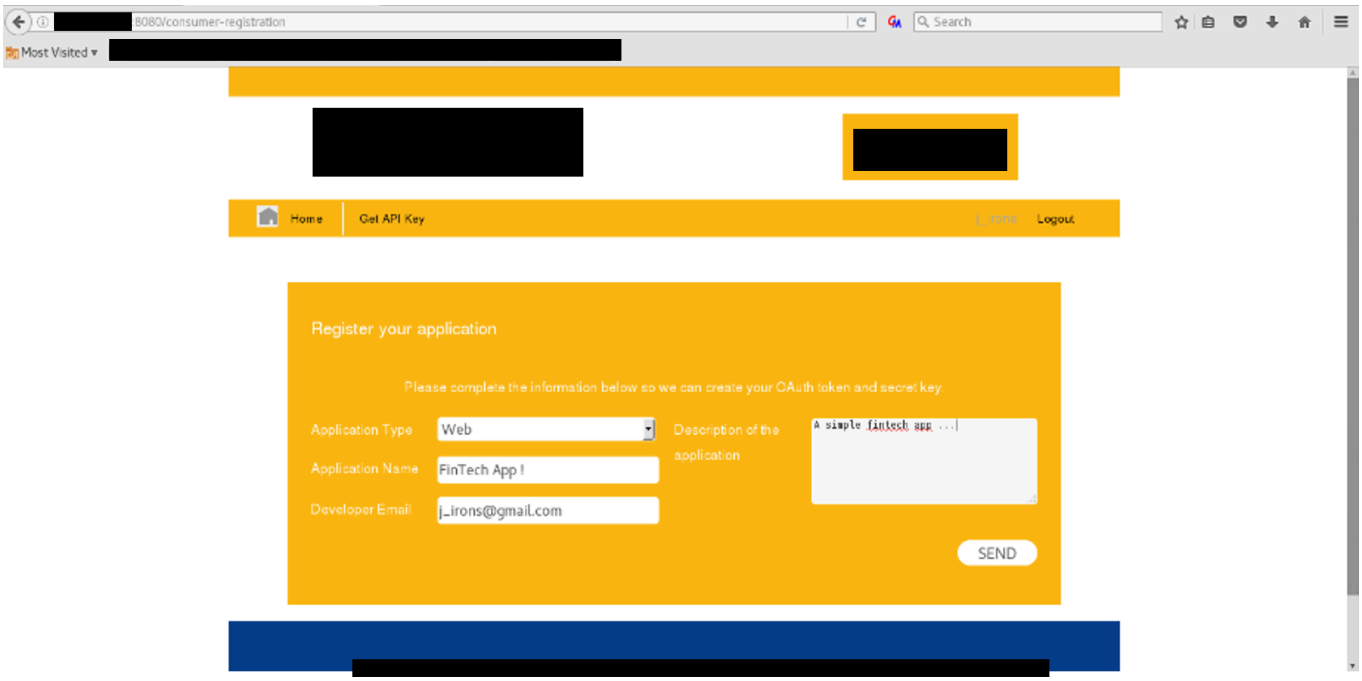
`CVSS 3.1 Score` : 5.4 (Medium)

`Description` : During our testing activities, we identified that the web page responsible for consumer registration is vulnerable to Cross-Site Request Forgery (CSRF) attacks.
Cross-Site Request Forgery (CSRF) is an attack where an attacker tricks the victim into loading a page that contains a malicious request. It is malicious in the sense that it inherits the identity and privileges of the victim to perform an undesired function on the victim's behalf, like change the victim's e-mail address, home address, or password, or purchase something. CSRF attacks generally target functions that cause a state change on the server but can also be used to access sensitive data.

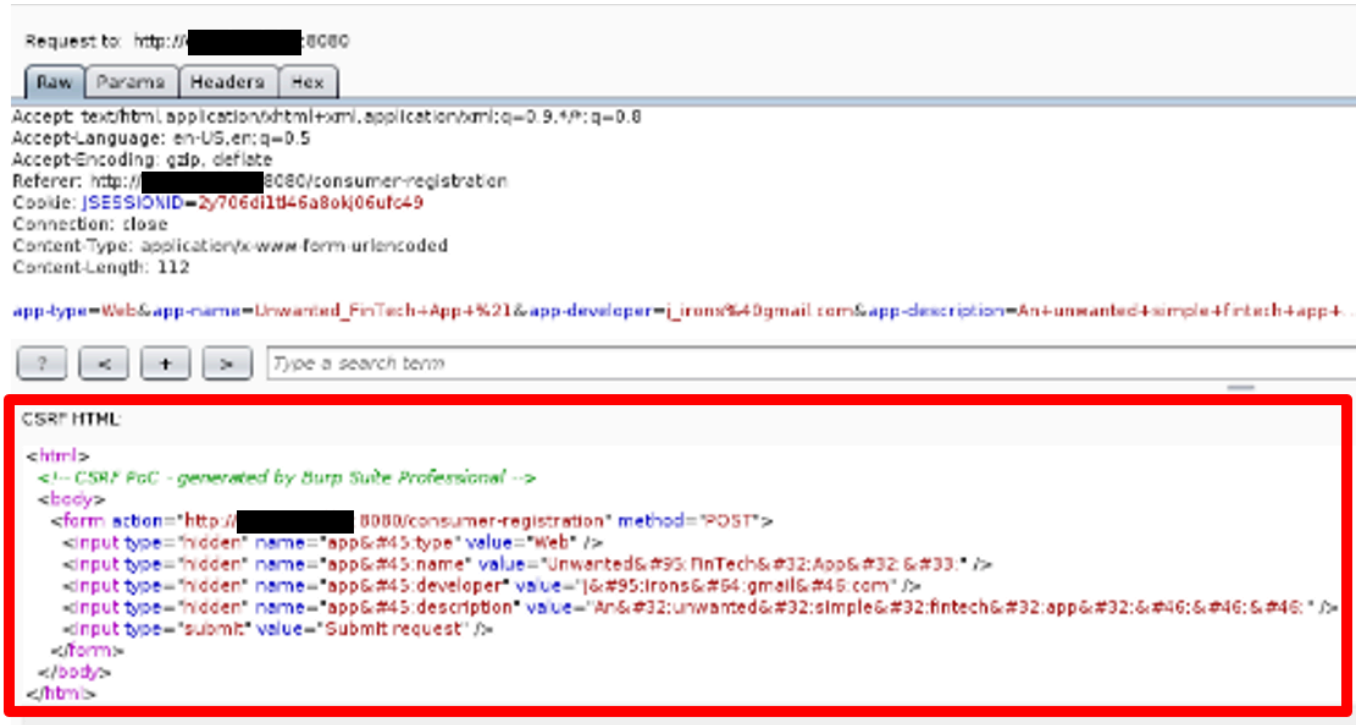
`Impact` : The impact of a CSRF flaw varies depending on the nature of the vulnerable functionality. An attacker could effectively perform any operations as the victim. Because the attacker has the victim's identity, the scope of CSRF is limited only by the victim's privileges. Specifically, an attacker can register a fintech application and create an API key as the victim in this case.

`POC` :

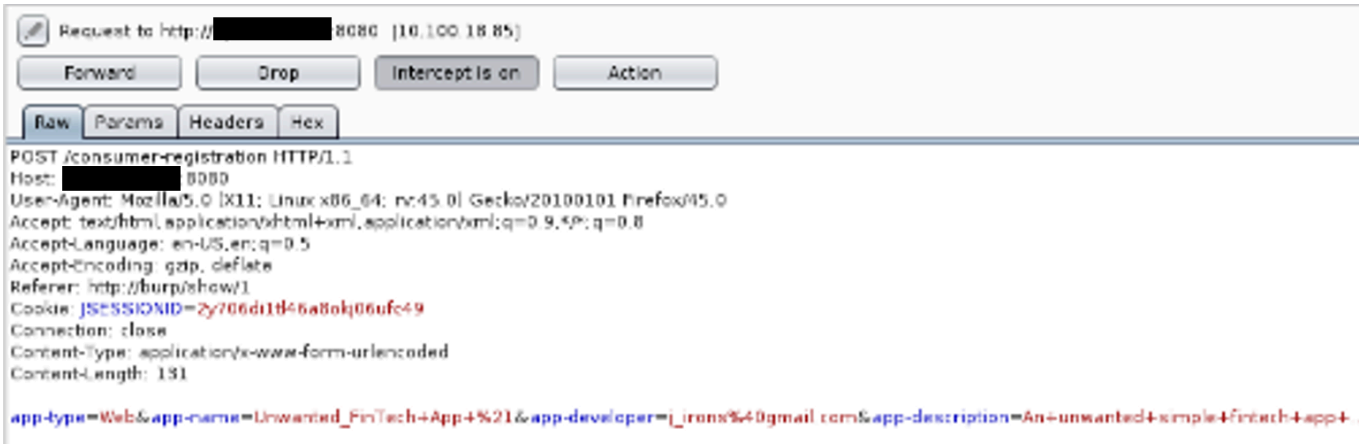
Step 1: Using an intercepting proxy, we looked into the request to create a new fintech application. We noticed no anti-CSRF protections being in place.



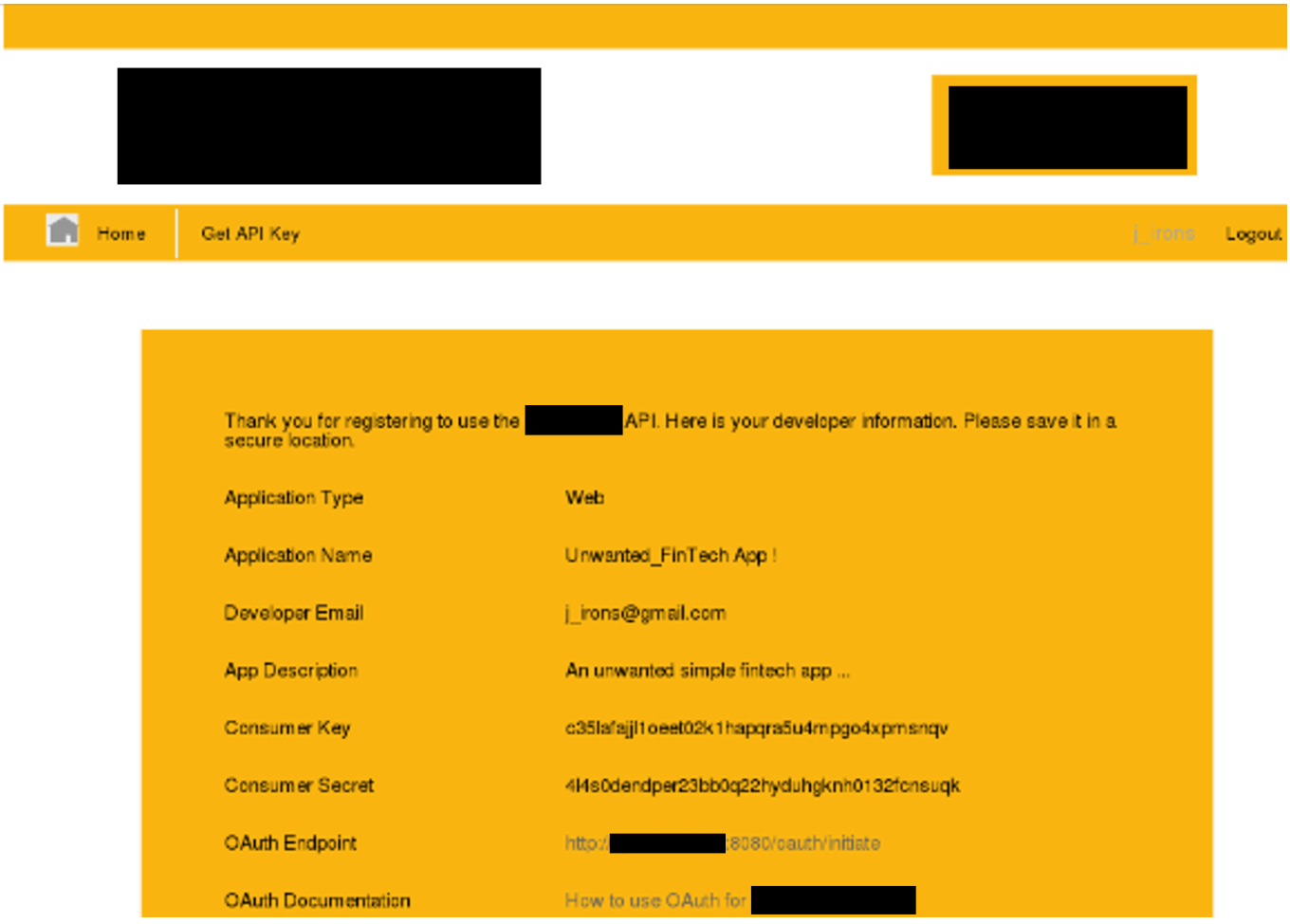
Step 2: We used the abovementioned request to craft a malicious HTML page that, if visited by a victim with an active session, a cross-site request will be performed, resulting in the advertent creation of an attacker-specific fintech application.



Step 3: To complete the attack, we would have to send our malicious web page to a victim having an open session. The following image displays the actual cross-site request that would be issued if the victim visited our malicious web page.



Step 4: The result would be the inadvertent creation of a new fintech application by the victim. It should be noted that this attack could have taken place in the background if combined with finding 6.1.1. <-- 6.1.1 was an XSS vulnerability.



CVSS Score Breakdown

`Attack Vector:`	Network - The attack can be mounted over the internet.
`Attack Complexity:`	Low - All the attacker has to do is trick a user that has an open session into visiting a malicious website.
`Privileges Required:`	None - The attacker needs no privileges to mount the attack.
`User Interaction:`	Required - The victim must click a crafted link provided by the attacker.
`Scope:`	Unchanged - Since the vulnerable component is the webserver and the impacted component is again the webserver.
`Confidentiality:`	Low - The attacker can create a fintech application and obtain limited information.
`Integrity:`	Low - The attacker can modify data (create an application) but limitedly and without seriously affecting the vulnerable component's integrity.
`Availability:`	None - The attacker cannot perform a denial-of-service through this CSRF attack.

Example 3: Reporting RCE

`Title`: IBM WebSphere Java Object Deserialization RCE

`CWE`: CWE-502: Deserialization of Untrusted Data

`CVSS 3.1 Score`: 9.8 (Critical)

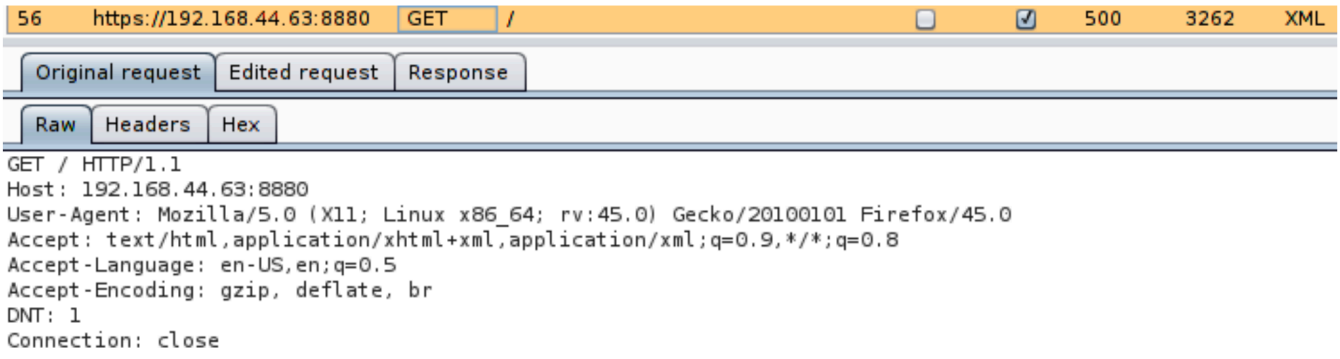
`Description`: During our testing activities, we identified that the remote WebSphere application server is affected by a vulnerability related to insecure Java object deserialization allowing remote attackers to execute arbitrary commands. By issuing a request to the remote WebSphere application server over HTTPS on port 8880, we identified the existence of raw, serialized Java objects that were base64-encoded. It is possible to identify base64 encoded serialized Java objects by the "rO0" header. We were able to craft a SOAP request containing a serialized Java object that can exploit the aforementioned vulnerability in the Apache Commons Collections (ACC) library used by the WebSphere application server. The crafted Java object contained a `ping` command to be executed by the affected system.

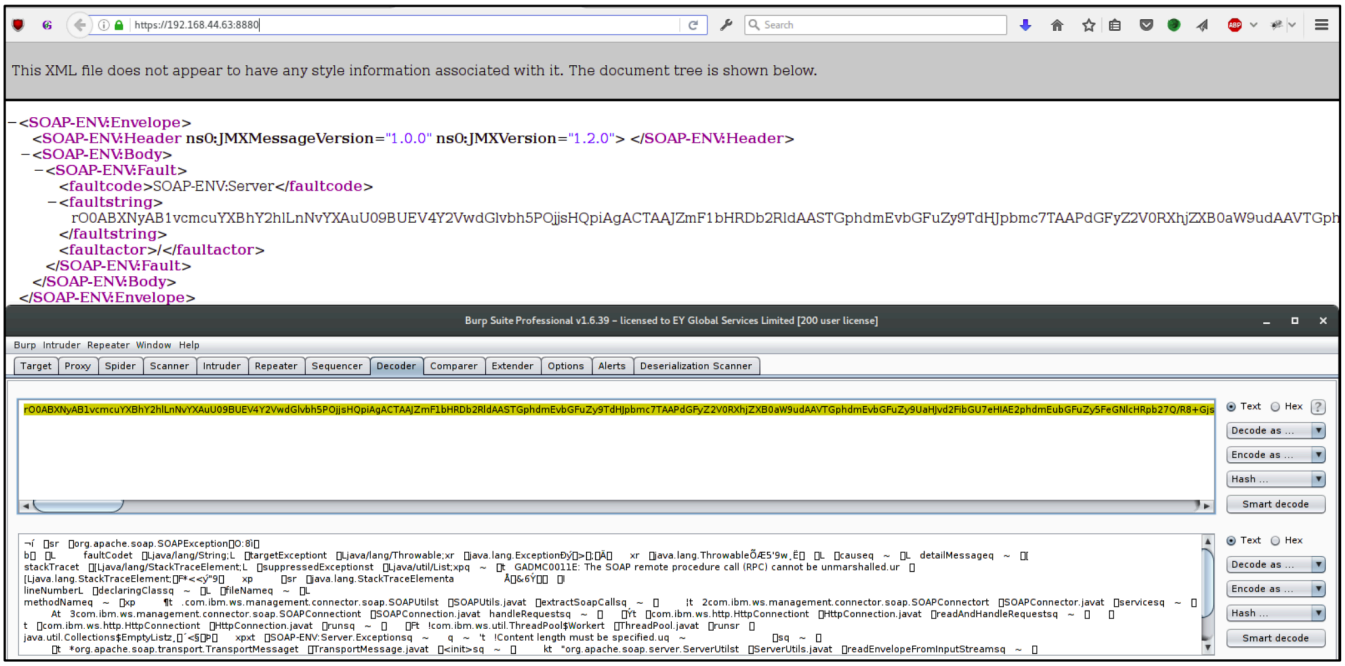
`Impact`: Command injection vulnerabilities typically occur when data enters the application from an untrusted source, such as a terminal or a network socket, without authenticating the

source, or the data is part of a string that is executed as a command by the application, again without validating the input against a predefined list of allowed commands, such as a whitelist. The application executes the provided command under the current user's security context. If the application is executed as a privileged user, administrative or driver interface, such as the SYSTEM account, it can potentially allow the complete takeover of the affected system.

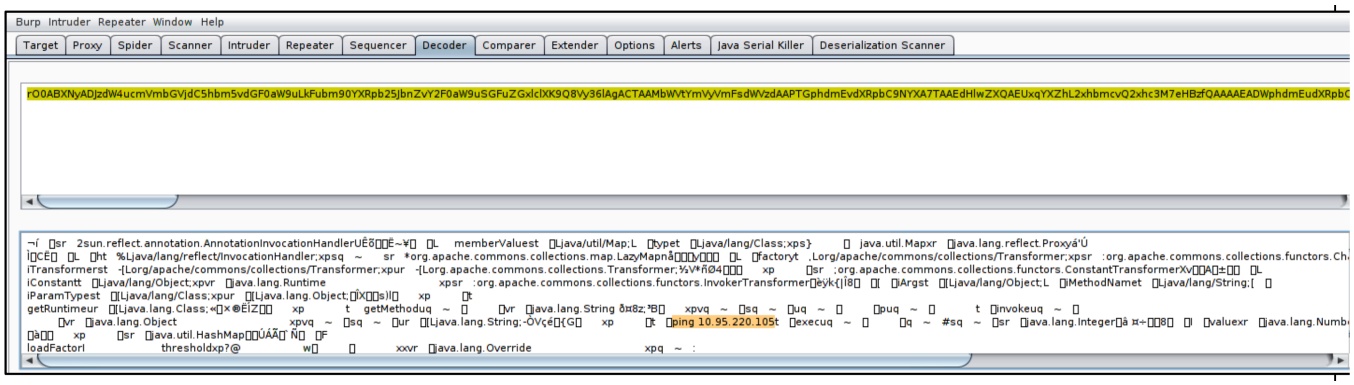
`POC` :

Step 1: We identified that the application uses serialized data objects by capturing and decoding a request to port 8880 of the server. The following images display the original request and the remote server's response, along with its decoded content.

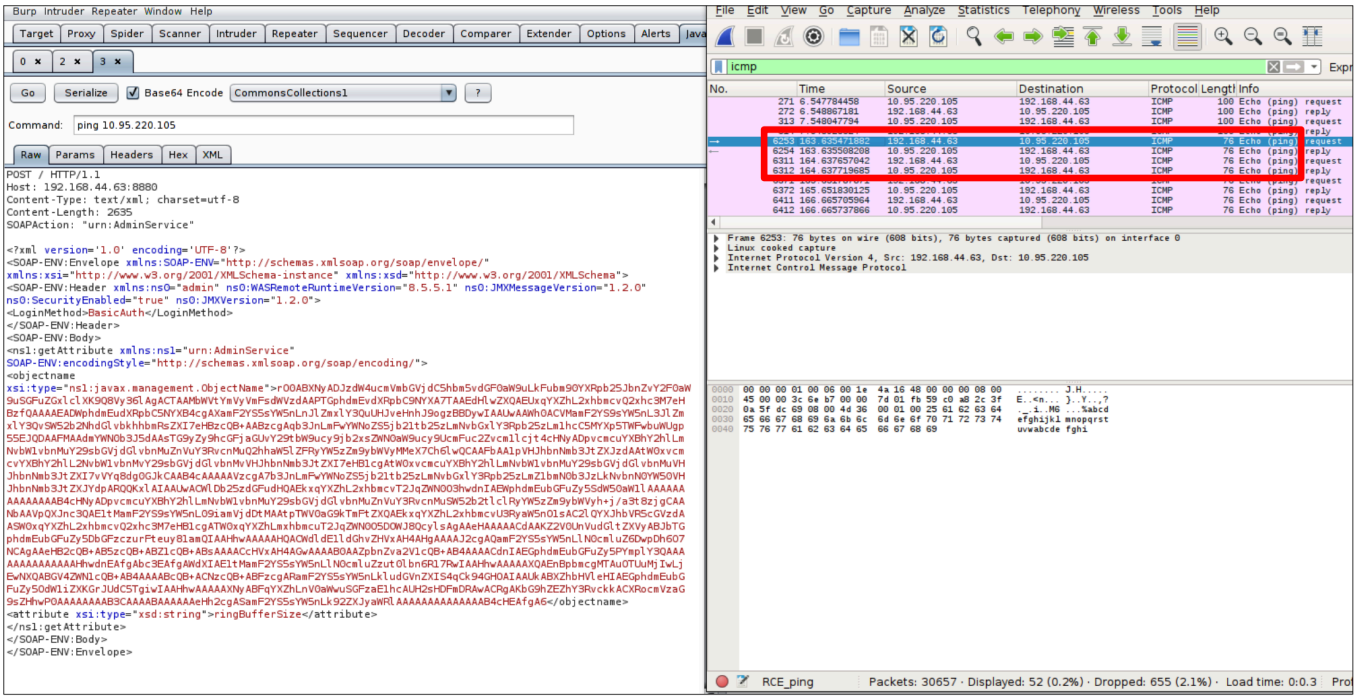




Step 2: We crafted a SOAP request containing a command to be executed by the remote server. The command would send `ping` messages from the affected server to our host. The image below displays the crafted request and its decoded payload.



Step 3: The following image displays the crafted SOAP request allowing to remotely execute a `ping` command from the affected system. Capturing traffic via Wireshark, we observed the `ping` request from the Websphere application server to our machine.



CVSS Score Breakdown

`Attack Vector:`	Network - The attack can be mounted over the internet.
`Attack Complexity:`	Low - All the attacker has to do is send a crafted request against the vulnerable application.
`Privileges Required:`	None - The attacker can be mounted from an unauthenticated perspective.

`User Interaction:`	None - No user interaction is required to exploit this vulnerability successfully.
`Scope:`	Unchanged - Since the vulnerable component is the webserver and the impacted component is again the webserver.
`Confidentiality:`	High - Successful exploitation of the vulnerability results in remote code execution, and attackers have total control over what information is obtained.
`Integrity:`	High - Successful exploitation of the vulnerability results in remote code execution. Attackers can modify all or critical data on the vulnerable component.
`Availability:`	High - Successful exploitation of the vulnerability results in remote code execution. Attackers can deny the service to users by powering the webserver off.