

Bugcrowd: Beginning with Bug Bounties

What is Bugcrowd?

Bugcrowd is a bug bounty platform that connects security researchers with companies hosting active bug bounty programs. Bugcrowd is fully integrated so everything from, rules of engagement, program scope, bug reporting, and payments are all done through the Bugcrowd platform. Bugcrowd is unique because it supports a plethora of choices for the scope of a program. Programs can support hardware devices, API endpoints, IP ranges, desktop or mobile applications, browser add-ons, and more.

To a Company:

Bugcrowd gives companies a structured and low friction platform to host their bug bounty program.

Companies can manage multiple programs from one account, for example, a company might want to have a public program with one scope, then include new targets in a private program. When creating a program the program managers can include the vulnerability severities they are paying out for as well as the payout range for those severities. Bugcrowd has recommendations for what bugs should fall under what severity category, but the ultimate decision is up to the program manager.

The program manager can review bug reports and communicate with the researcher through that report. If more information needs to be collected, the program manager requests that from the researcher. Bugcrowd can act as an intermediary between the researcher and the company that they are reporting with. Program managers can also change the status of a bug depending on what stage of the bug processing they are in (Triage, Fixed, Duplicate, Not Fixing, Invalid Bug).

On bug reports, Bugcrowd includes references and remediation techniques for the bug that has been reported, this is geared towards improving the quality of life of less security-focused developers that may be using the platform.

Program managers can payout researchers on the Bugcrowd platform, and recommended payout amounts are available, however, the ultimate decision is up to the company.

Companies have the ability to see metrics on their various programs including, how many bugs have been reported, the severity distribution of the bugs, the types of bugs that seem to be recurring, and more. This can help a company identify weak points in their development practices and better train their engineers to cover those bugs.

Companies have the option to include integrations into their program so that tickets can be introduced into their development cycle when a bug is triaged. Integrations can be bi-directional so that when a ticket is resolved in the integration it will be pushed to the Bugcrowd platform.

To a Researcher:

As a researcher, Bugcrowd is a marketplace for bug bounty programs. Researchers are free to select any program they qualify for and investigate that target right away. Companies provide detailed descriptions of the scope of their program and many include a transparent rewards system for the bugs researchers might find. On the program description page companies can outline specific areas they would like researchers to focus on. This can help researchers find programs that they see their skills best apply to.

When submitting a bug report researchers are able to provide detailed descriptions of their findings and include supporting documents in their report. In addition, when specifying the vulnerability type of a bug, the researcher can see the severity rating that Bugcrowd (or the Company) has given that bug type.

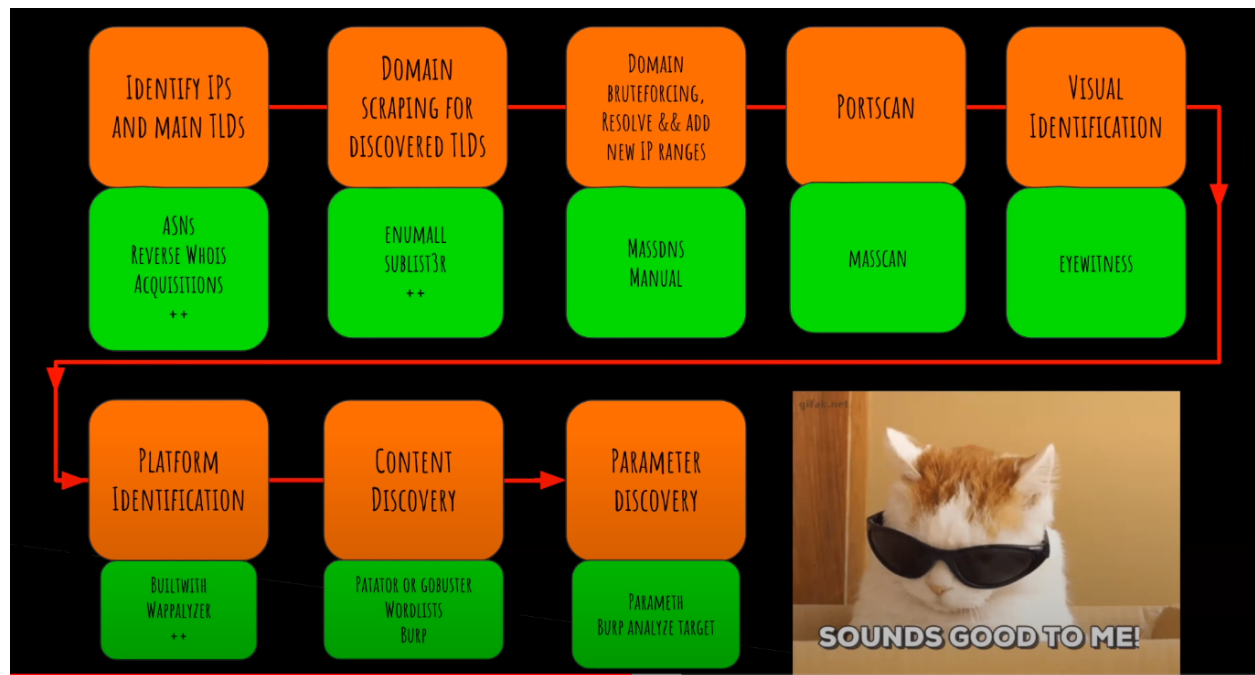
Researchers are not able to disclose any details of their finds to the public unless given written permission by the company responsible for the bug.

Researchers are paid directly through Bugcrowd.

Pentesting / Vulnerability Assessment

This section's goal is to describe the pentesting / vulnerability assessment life cycle and mention tools that go along with those sections.

Mapping/Discovery



Recon

Let's discover what exists under the scope of the program. What domains are out there, what subdomains exist for the target? What IP ranges do they own? This is a huge part of taking part in bug bounties, finding a subdomain that no one else has found gives you first pick at low-hanging fruit and gives you unique avenues to find bugs on a program.

Acquisitions

Sometimes the scope of programs can include all domains owned by a program, a lot of the time this can include acquisitions.

- [CrunchBase](#) - business information portal, includes the acquisitions of a company

ASN/IP Discovery

all of these are for IP discovery find out what the IP range a company owns

- **Hurricane Electric Internet Services**
- [ARIN](#)
- [RIPE](#)
- **REVERSE.REPORT**

Linked Discovery

- [gospider](#) - cli spider for parsing robots.txt, js files, and returns status codes
- [hakrawler](#) - command line spider, supports recursion

- 1) TURN OFF PASSIVE SCANNING
- 2) SET FORMS AUTO TO SUBMIT (IF YOU'RE FEELING FRISKY)
- 3) SET SCOPE TO ADVANCED CONTROL AND USE STRING OF TARGET NAME (NOT A NORMAL FQDN)
- 4) WALK+BROWSE, THEN SPIDER ALL HOSTS RECURSIVELY!
- 5) PROFIT (MORE TARGETS)!

URL Discovery

We want to have as large of a target range as possible. The more TLDs we have for a given company the more targets we might have to attack. All depending on scope of course.

- [amass](#) - can be used to search for an asn and get all the TLDs associated with it
- [whoxy.com](#) - performs a reverse whois search on a domain listing all the domains that they owned in the past
- [domlink](#) - reverse whois look up tool, find the whois information on target and look for other registered domains with the same whois info. uses whoxy api key
- [builtwith](#) - use google analytics tags to identify other websites using the same tag
- [google](#) - quote the copyright tag for the main website and you might be able to find more seed domains
- [shodan](#) - very verbose internet spider

Subdomain Discovery

- [sublist3r](#) - subdomain scraping with using different sources including, google, bing, crt.sh, ask, dnsdumpster, virustotal
- [brutesubs](#) - docker based subdomain scanner using multiple scanners
- [amass](#) - subdomain discovery tool, includes reverse DNS methods, includes permutation discovery (i.e. dev-1.xxx.com, dev-2.xxx.com) (OWASP PROJECT) **FASTEST**, use this and subfinder
- [subfinder](#) - similar to amass, json output, multi resolver for bruteforce (**also very fast**)
- [subdomainizer](#) - detects api keys, finds subdomains referenced in js files, finds cloud services in js files
- [subscraper](#) - js subdomain scraper, supports recursion
- [github-subdomains.py](#) - searched github for a domain name and then will keep track of subdomains
- [shosubgo](#) - scraping subdomains on shodan
- [cloud scanning](#) - query a ip address on port 443 then check it's ssl certificate to see if it contains any key words for the target (advanced technique, not many tools \$\$)
- [bufferover.run](#) - tool for cloud scanning <http://tls.bufferover.run/dns?q=twitter.com> change query parameter to target tld

Subdomain Brute Forcing

- [massdns](#) - brute forces subdomain in a given dictionary **FASTEST**
- [gobuster](#) - gobuster can be used to bruteforce subdomains
- [all.txt wordlist](#) - jhaddix all subdomain wordlist
- [amass](#) - yes amass can do it all in “enum” mode it can bruteforce subdomains and it can specify any number of DNS resolvers (more resolvers means more faster)
- [shuffleDNS](#) - built on mass dns core and can take wildcard characters
- [commonspeak2-wordlist](#) - wordlist leveraging Google BigQuery data to generate content discovery wordlist
- [commonspeak2](#) - Leverage Google BigQuery to create wordlists

Auxiliary Subdomain Discovery

- **DNSSEC/NSEC**
- **Github Recon** - search for domains listed in the orgs public github content
- **Google Dorking**

Enumeration

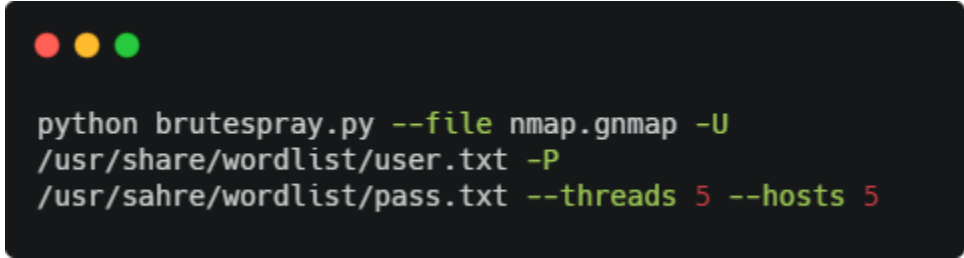
Enumeration consists of narrowing down the things we found in the discovery phase, okay we have a ton of IPs, now let's check the ports that are open.

Port Scanning

- [nmap](#) - scan ports, can see software version, identify open/closed ports
- [masscan](#) - super fast port scanner written in c, use this to find open IP addresses then use the features of nmap to find more about these open ports. Needs the iplist not domains **FASTEST**
- [dnmasscan](#) - dns resolution on domain names for ip address

Credential Brute Forcing

- [brutespray](#) - sprays credentials at services like ssh



```
python brutespray.py --file nmap.gnmap -U  
/usr/share/wordlist/user.txt -P  
/usr/share/wordlist/pass.txt --threads 5 --hosts 5
```

Github Dorking

- [jhaddix-script](#) - dork github for leaked credentials or repos, employees might upload their own code that includes something they did at work
- [github-search](#) - find passwords, api keys, and sensitive data exposure in github repos

Visual Identification

- [eyewitness](#) - takes screenshots of all pages that are in a list of domains, can take a picture for http and https pages
- **httpscreenshot**

Wayback Enumeration

- [archive.org](#) - look for pages that might have had credentials on it and check that they were not once exposed, try this if you hit a 401 or 403 error that just requires basic auth.
- [waybackurls](#) - fetches all the URLs that the Wayback Machine knows about for a domain

Directory Brute Forcing

- [gobuster](#) - subdirectory brute forcer written in go
- [ffuf](#) - subdirectory brute forcer written in go
- [content_discovery_all.txt](#) - massive wordlist including all raft wordlists

Platform Identification

- **wapplyser** - identify the tech stack on any website
- **builtwith** - find out what a website is builtwith
- **retire.js (burp extension)** - identifies out of date JS libraries running on a website
- [burp-vulners-scanner](#) - scans the Server header tag for server version number and looks up CVEs against that version.

Auxiliary

- **WPScan** - determines plugins and users for a wordpress install, also looks up vulns for the plugins installed
- **CMSmap**

Parameter Brute Forcing

- **parameth** - brute forces possible parameters for a request
(Portswigger/bachslash-powered-scanner) <- github wordlist go to /resources/params

Javascript Parsers

- [ZAP AJAX Spider](#) - spiders site with headless site, helps get ajax code
- [JSParser](#) - finds urls in JS (not as good as linkfinder)
- [Linkfinder](#) - gets the urls in JS files

Organization

The above steps result in a ton of data, especially if the scope is large. Organizing this data so that you can systematically explore interesting information is important. Keeping things organized can also reduce the amount of rework that you do.

- **xmind** - mapping software that can include files, links. Map tools to output and mark things in color to indicate completeness of a task. (premium)

Looting

Vulnerability Analysis

XSS

XSS - Cross Site Scripting

- **polyglots** - injection string that executes in multiple different contexts and could possibly bypass filters

Blind/Out of Band XSS

- [xsshunter](#) - better payload generator
- **sleepypuppy** - open source netflix tool to attempt blind xss (deprecated)

Server Side Template Injection

- **tplmap** - template injection mapping tool

SSRF (Server Side Request Forgery)

- **ssrf bible**
- [cloud_data.txt](#)

Code Injection / Command Injection

- **commix** - command injection
- **backshash-powered-scanner** - automates sending ` and \ to endpoints to see if escape characters executed disallowed characters

Misconfiguration

- [s3scanner](#) - find open buckets on orgs and dump their contents

Subdomain Takeover

- [can-i-take-over-xyz](#) - list of service providers and whether or not a subdomain can be taken over given the provider.

- [nuclei](#) - scanning framework, can find api keys, subdomain takeover etc
- [aquatone](#) - subdomain discovery, scanning, and takeover
 - After identifying domains, even if aquatone says none are vulnerable consider using [eyewitness](#) to look for pages that return 404, they might still be vulnerable to subdomain takeovers
- [takeover.cyberint.com](#) - I am not sure about this one, it should be as simple as typing in the domain name, but in my experience it takes > 24hrs to get results

Flashbang - give it a swf file and it will return the possible xss payloads

Liffy - tool that tries to find local file inclusion.

Burpy - checks all the logs in burp and checks for CSRF

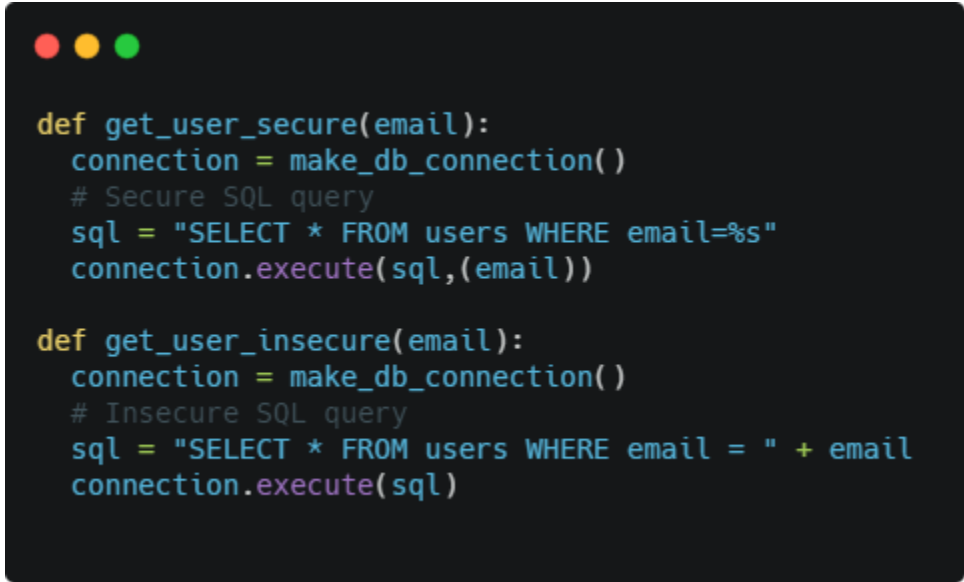
Exploitation

Reporting

OWASP Top 10

Injection

SQL Injection



```
def get_user_secure(email):  
    connection = make_db_connection()  
    # Secure SQL query  
    sql = "SELECT * FROM users WHERE email=%s"  
    connection.execute(sql,(email))  
  
def get_user_insecure(email):  
    connection = make_db_connection()  
    # Insecure SQL query  
    sql = "SELECT * FROM users WHERE email = " + email  
    connection.execute(sql)
```

SQL Injection can allow attackers to execute SQL queries on the database the insecure query is calling to. This is because attackers can escape out of queries and write their own. Attackers can dump databases, delete databases, enter new records into the database and edit records.

Prevention

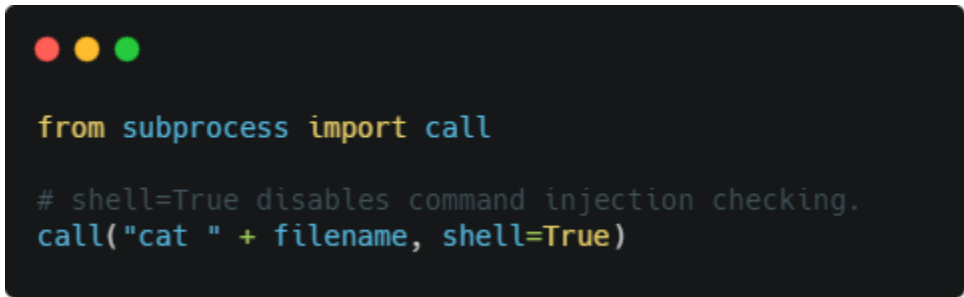
In order to prevent SQL injection it is important to sanitize input and even more so use well vetted database drivers and pass sql parameters into their query functions and not the query itself. If user input is passed as a parameter to database driver functions they will sanitize the input.

An even simpler way is to use a ORM (Object Relational Mapping) framework so that you do not need to write many SQL queries.

If you find yourself concatenating strings together in a query you are probably doing it wrong!

Command Injection

Command injection, like SQL injection, is when the server side machine is tricked into executing commands that it should not run. In the case of command injection this is usually because user input is being passed as an argument to a command line script and it is not being sanitized properly. This can lead to command execution by escaping out of the control sequence and running the attackers command.



```
from subprocess import call

# shell=True disables command injection checking.
call("cat " + filename, shell=True)
```

The following function would be vulnerable to command injection because if the attacker were to pass in ` && ls ` the `ls` command would also execute.

Prevention

Like they told me in health class the best prevention is abstinence. Try to avoid using command line calls and use secure API calls instead. Also **sanitize** your inputs, make sure values like `;`, `&`, `\`, `|` and `` are scrubbed from the input.

If you have to use command line commands you should at least run the process at highly restricted permissions so that attackers will not be able to see anything on the machine or execute commands that require any level of permission.

Broken Authentication

Password Mismanagement

Password mismanagement can come in many forms, it can be how the users of an individual treat their passwords or it can be how an org manages their own users passwords.

Organizations should always store hashed and salted passwords and ensure that users are using complex passwords. Password resets should send links to the users email, but the link should expire within a reasonable amount of time.

If a user is already logged onto your service make sure that they need to enter their old password before they can reset it.

Don't prevent password brute-forcing with a lockout, this will make it a bad UX if an attacker were to ever try and gain access to a user's account. Instead opt for CAPTCHA login, this will evade scripting.

User Enumeration

User enumeration can allow attackers to gather a list of users that are registered with a site. If the attacker can figure out the valid usernames of your users then they have half of the auth for those users. Even worse, if emails can be used to log in then the attacker can target those individuals with phishing emails to try and leak their password.

User enumeration targets can be login pages, password reset pages, and profile urls.

While this exploit is easy to use, and fairly common the impact is relatively small on an org.

Prevention

When a user is logging in and fails to authenticate, return a generic error message that does not give away the details of the credentials the user tried. Using phrases like "incorrect password" gives away that the user account exists. Instead opt for something like "that username and password combination does not exist". This is ambiguous because it does not say that the username exists, just that the combination of the two does not exist. This also applies to password reset pages. If an attacker can tell if a username exists from two different messages

Even with generic error messages, some attackers might notice that responses from requests with valid usernames might take longer to return than those that do not exist, be sure that all code paths have a similar average runtime to prevent attackers from exploiting these time-based attacks.

Ensure that the HTTP response headers are the same when a user exists and doesn't exist if they failed to authenticate.

A method to prevent brute forcing is **exponential backoff**; this is where the request takes longer to respond after every failed login attempt.

Session Fixation

Sessions are used so that returning users can leave then come back to a website without having to reauthenticate. This is done by exchanging a session token between the server and the browser. Mismanagement of these session tokens can lead to broken authentication of accounts. If an attacker gets a session token they can impersonate the actual owner of that session token. One way attackers can get the session token of another user is through **session fixation**, this is when the session id given in a post request at login is then used to manage that session.

Prevention

In order to prevent session fixation, orgs should reassign session keys at login. The systems should also only allow for system-generated cookies to be used at session IDs, and old sessions should time out/expire.

When a user logs out of the service the server should render the session cookie obsolete.

Weak Session IDs

Easily guessable session IDs can ruin auth. If an attacker is able to decode session ids or enumerate them they are virtually useless.

Prevention

In order to prevent weak session IDs, you can use any modern frameworks session management, and set tamper-proof cookies to true. Tampering with the session id cookie will tell the server that it should invalidate the session.

Sensitive Data Exposure

Unencrypted Communication

Insufficient encryption can lead to man in the middle attacks.

Prevention

Enable HTTPS on your login pages, buy a certificate and configure it on your webserver. Also set the cookies to secure so that sessions can only be established if the session is over HTTPS.

XML External Entities (XXE)

XXE can be dangerous because it may allow attackers to read local files by exploiting XML parsers. If a site accepts XML it could allow attackers to execute XXE. Attackers can leverage this attack by creating their own XML file, hosting the file on their own server, then referencing the XML file for the website to parse.

Prevention

Limit the permissions on the webserver process so that it does not have permission to read and execute files out of scope of its functions. In addition disable inline DTDs.

Broken Access Control

General

Broken access control usually occurs when an org takes the “security through obscurity” approach. There might be files and links that normal users should never be allowed to visit, but perhaps they have predictable names. If a user that is not supposed to see these pages can discover the link and use them it is broken access control.

Prevention

Preventing broken access control comes down to three key strategies:

- **Authentication** - the users needs to be correctly identifies, is the admin really an admin
- **Authorization** - Okay so this is user X, should they be able to perform this action
- **Permission** - Checking - Is a user allowed to see this file

Directory Traversal

URL paths can sometimes decide what files are accessed by the user. Orgs need to be sure that users cannot access sensitive files on the server by inputting paths that should not be accessible. Using the ‘../’ syntax might allow attackers to explore files on your server.

Prevention

Using a third party CMS to manage files protect against directory traversal by implementing filters and path sanitation.

Hosting data on another server could also help mitigate the risks of directory traversal, if sensitive data and public data are segmented then it should be harder for attackers to exploit the sensitive information.

Again, run with restrictive permissions. Don’t allow the web application to read files outside of what it needs and then attackers will not be able to see them either. While this does not stop directory traversal, it does mitigate the risks.

Security Misconfigurations

Lax Security Settings

Most applications security configurations are fairly lax out of the box, for example they have default credentials, debugging turned on, and do not enforce HTTPS. These security misconfigurations could allow for attack escalation by a hacker. Pre-production environments are also configured in a way that allows programmers to get a lot of information about a system so that they can better understand how

their code is working/performing/failing. This is not information that a customer should see and can leak details about the infrastructure of an org.

Prevention

In order to lax security configurations from reaching the production environment it is a good idea to automate the build process so that security settings are automatically changed to predetermined values before code hits production servers. Also make sure to review new software components and clearly separate code files and configuration files.

Certain accounts should be created to do certain tasks, not all the developers should have god mode all the time if they never need to do escalated tasks.

Information Leakage

Information in HTTP headers can leak information about the software stack a website is using. They can tell the exact server software and version number the system is running and even leak components by using specific parameter names for that component. Error messages can also leak information, if there is a 500 internal, make sure that it is scrubbed of any sensitive information before serving it to the end user.

Prevention

Be sure to keep separate directories for public information and configuration files.

Clean urls to avoid adding telling extensions 'jsp', 'php', 'asp'.

Cross-Site Scripting (XSS)

Reflected XSS

Reflected XSS is when an application includes unescaped user input as a part of the HTML output. This can allow for the crafting of malicious links. Think about a search bar. If I find an XSS in a search bar I am only going to be able to see it, it is not going to persist. This form of XSS can be used to phish session cookies.

Stored XSS

Stored XSS is persistent, such as posting a XSS as a blog post and having that code run on other peoples machines if they visit the post. This is considered a critical vulnerability.

DOM XSS

???

Insecure Deserialization

Insecure deserialization can occur when an attacker modifies the application logic in order to gain access to a system that he or she should not have otherwise. Serialized objects might contain information such as access level and can easily be changed by hackers.

Prevention

The only surefire way to prevent insecure deserialization is to not accept serialized objects from untrusted sources. However if this is not an option, be sure that the application is implementing integrity checking and enforcing strict type constraints on deserialized objects.

Using Components with Known Vulnerabilities

Insufficient Logging & Monitoring

Common Vulns

Server Side Template Injection