A hunter’s methodology V5.0

The rat is out of the bag as they say. We all know I love main application hacking because it precludes broad scope hacking but how exactly do you go about this?

First of all, I am going to refer to

Setting Up Burp Suite

This article will contain the basis for my vulnerability and bug hunting. I always start with doing this before even exploring the application. But what do I explore and click?

1. Exploration and enumeration

First of all, I want to make sure I have clicked on every button, link, and endpoint that I can find but even before I do that, I read the manual. Yes, it might seem strange but the manuals are goldmines, my friends. They tell you exactly what you can and can not do and in this war of today, information is power!

Yes, you are right, we are at war. A cybercriminal is also on a warpath and so should we be with the big difference that we don’t have to go to jail for it! We all know that no good wars are even won without good preparation and a well-planned out strategy but before you can even start planning, you need information in the form of documentation, manuals, and FAQ pages. Make sure you open every single piece of functionality the manual speaks of and skip no steps.

Especially if the manual tells me I can not do something security related, I will be sure to look into that. You’d be surprised at the dumb stuff you can find by doing this while others simply skip it.

I will also try to investigate the JS files to look for any potential endpoints I might have missed and even consult waybackmachine (yes the actual website, not the tool). This all serves a purpose as I want to get my site tree in burp suite as full as possible.

Whenever you register though, make sure to enter the following text into every single input field where it is allowed:

```
‘”`><img src=x>${7*7}
```

This will test for:

- SQLi
- CSTI
- SSTI
- XSS in 3 different contexts

And the beauty is that if you enter this for a name for example, and that name is used later on in the application, you are auto-testing those parts of the application. This is the lazy method though and might miss a lot.

1. Investigating parameters

Now that my site tree is filled up, I want to start with investigating the parameters on their own, I can easily do this by going into burp and filtering to only show my parameterized requests **but remember this will not show parameters that are part of the URL (For example /invoices/123 contains a parameter 123 which is an identifier. THIS WILL BE FILTERED OUT)**. I will have specific things I am looking out for and they have to do with a base level knowledge of most exploits.

So what am I looking for? First and foremost, anything business logic related. Dates I can mess with where I should not, Ordering negative quantities of items, Re-applying coupons, … anything I can think of that is related to the functionality of the site and that is why it is so important to get to know it first. I do not joke when I say that it can take me up to a week to get very well acquainted with a system.

You never want to dive straight into hacking because hacking means you use something in a way it is not intended to be used but you have to know the intended use first. Besides, it’s easier to automate some of the more technical vulnerabilities while the more functional ones like business logic are much harder. This is an advantage for us.

2. What are we looking for?

Identifiers

For example, id=1 or invoiceId=123 - We are testing for IDORs here. Try changing the identifiers. Make sure you look for the more obscure identifiers, for example by looking into files as well or URLs.

Sometimes you might encounter a UUID which seems like a random string of numbers and letters but don't be discouraged, you can still look for IDORs but make sure you can also find an endpoint that lists all the identifiers, for example, "GET /users" might show all the userIDs if that is a UUID.

https://www.youtube.com/watch?v=hN19l9T7Uws&list=PLd92v1QxPOpqFO4bcV3tGEJgPsNtJycDV

Privileged endpoints

Sometimes we can find endpoints such as /admin/getUsers.php or even when we are just logged out, we might be able to play with the requests that require you to log in to the application. Of course, when you are not logged in, you should not be able to send requests successfully when they require authentication.

We have to test any endpoints that require a login but we also have to test any admin/higher privilege user's endpoints with a lower privilege level user.

https://www.youtube.com/watch?v=wCqKY3iHXuc&list=PLd92v1QxPOppEqAGjWmAkUehr5dIUIj5g

CSRF tokens

A CSRF token is a protection mechanism that is meant to prevent attackers from simply being able to copy the input forms you have and hosting them on their own site while making a request to your site. This might not seem so dangerous at first but imagine if someone could trick you into clicking a button that says anything and after clicking that, they change your email and before you know it, you will have lost your account with no way to recover!

Here are some common CSRF bypasses:

- Check if the token is present on any form it should be — ONLY Create, Update and Delete forms after a login page should have a CSRF token
- Server checks if the token length is correct
- Server checks if the parameter is there
- The server accepts an empty parameter
- The server accepts responses without a CSRF token
- The token is not session bound

You can test these by changing the CSRF parameter in a request and seeing if it still gets accepted.

https://www.youtube.com/watch?v=ImqLlFMQrwQ&list=PLd92v1QxPOppWJoLDAvH4uAgOj3IrqEOU

URLs

Whenever I see a URL, several alarm bells go ringing, I can see potential for several vulnerability types, for example LFI and RFI come to mind but also SSRF.

https://www.youtube.com/watch?v=Y_RNGl5snKM

https://www.youtube.com/watch?v=LF8s-x1QMlY

Here are some common LFI/RFI bypasses:

- Using // to bypass
- /\ to bypass
- \\
- %00 to bypass (null byte)
- @ to bypass
- URL encoding
- double encodings

For SSRF check the following:

- SSRF against server itself

- SSRF against other servers on the network

I might also try and look for open redirects. Here are some common bypassed for that:

- evil.com/expected.com
- Javascript openRedirects
- Hidden link open redirects
- Using // to bypass
- https:evil.com (browser might correct this, filter might not catch it)
- /\ to bypass
- %00 to bypass (null byte)
- @ to bypass
- Parameter pollution (adding the same parameter twice)

https://www.youtube.com/watch?v=qpgzTuBHGPA&t=77s

Uploading of images - SVG

SVG images are basically just XML files that describe what the image should look like. As you may know, XML opens the door for potential XXE vulnerabilities.

https://www.youtube.com/watch?v=MlJMeAobSxA&list=PLd92v1QxPOpq80iuGE0woKTxGNVdO_fFg

- Check: SVG files (images), DOCX/XLSX, SOAP, anything XML that renders
- Actions: Blind SSRF, file exfiltration, command exec

SOAP Requests

Since a soap request is nothing more than an XML request in it’s core, this is the ideal candidate to go looking for XXE possibilities.

https://www.youtube.com/watch?v=MlJMeAobSxA&list=PLd92v1QxPOpq80iuGE0woKTxGNVdO_fFg

- Check: SVG files (images), DOCX/XLSX, SOAP, anything XML that renders
- Actions: Blind SSRF, file exfiltration, command exec

Potential queries

I am by far no expert in SQLi but I will still look for it on every parameter of which I suspect it is used in query and I will test for verbose SQLi but also blind SQLi. I don’t have many tricks for you here friends, just the following:

- ‘“ to trigger an SQLi error in every parameter
- Run SQLmap to finish the PoC

JWT tokens

JWT tokens are very important authorisation mechanisms and they should be checked thoroughly. I recommend at least trying the following:

- Checking if None-signing algorithm is allowed
- Checking if Secret is leaked somewhere
- Checking if Server never checks secret (aka we can enter any secret and it will be accepted)
- Checking if Secret is easily guessable or brute-forceable

https://www.youtube.com/watch?v=_gmcIi6LVCg

Unmapped object properties

This might seem like a strange one but something, a user can have a property “isAdmin” and we can see this when we request GET /userSettings.php for example but there is no parameter in the application to edit this. Well, my friends, there is nothing stopping us from adding these kinds of properties to our POST /userSettings.php or PUT /userSettings.php requests.

XSS

I would not be called the XSS Rat if I did not have some tips for you here. First of all, in every single input field, try to get reflection of your value on the page. If you get reflection, check in what context it is reflected and craft your attack vector based on this.

- ‘“`><img src=x> into every input field, the moment you register and start using the application

- Enter a random value into every parameter and look for reflection
- See what context reflection is in
- Craft attack vector based on context
  — JS
  — HTML
  — HTML tag attribute
  — …
  — Url encode
  — HTML entities
  — Capital letters
  — BASE64 encode payload
- CSP might be active
  — Try bypasses
  — See what is active and where script can be gotten from
  — Encode them in base64
  — Mascarade script as data

https://www.youtube.com/watch?v=5r4E4EJwNo0&t=1s

Admin panels

Whenever I see a login panel or an admin panel, the first thing I will try to do is content discovery and try to find an endpoint that is not protected by the login screen. This will sometimes happen on custom authentication modules.

- Try headers on 403 pages

```
X-Originating-IP: 127.0.0.1
X-Forwarded-For: 127.0.0.1
X-Forwarded: 127.0.0.1
Forwarded-For: 127.0.0.1
X-Remote-IP: 127.0.0.1
X-Remote-Addr: 127.0.0.1
X-ProxyUser-Ip: 127.0.0.1
X-Original-URL: 127.0.0.1
Client-IP: 127.0.0.1
True-Client-IP: 127.0.0.1
```

`Cluster-Client-IP: 127.0.0.1`

`X-ProxyUser-Ip: 127.0.0.1`

`Host: localhost`

- Easy username/pass
- Directory brute forcing for unprotected pages
- Content discovery

Template injections (CSTI/SSTI)

Template injections are becoming more and more prevalent and we can easily check for them by looking at the versions of the templating engines or just entering ${7*7} into every parameter and if you get something like 7777777 or 49, you know you might have a template injection and you need to look into this.

https://www.youtube.com/watch?v=thUHg-2ci4E

- ${7*7}
- If resolves, what templating engine
- Try exploit by looking at manuals
  — URL encode special chars ({}*)
  — HTML entities
  — Double encodings

Captcha bypasses

To bypass captcha's I also have a few general tips for you. We all know that annoying captcha thing that pops up every now and again, well it has gone through many phases and all of them have known possible bypasses:

- Try to change request method
- Remove the captcha param from the request
- leave param empty
- Fill in random value

x. General tips

- Read the JS files! I can not stress this enough!
- It takes more than 8 hours to get to know an application well. Save the hacking for when you know what you are doing or supposed to be doing.
- Waybackmachine is not the only internet archive
- You can look for smaller bug bounty platforms such as bugbounty.jp for example
- If you don’t find anything, keep looking! It takes on average over 400 hours for a hunter to find their first bug.
- Don’t panic, bug bounties is not for money and not for finding bugs. It’s to have fun. To do what YOU want to do. Not being bound by coverage guarantees, just freedom. So cool!