Enumerating Esoteric Attack Surfaces

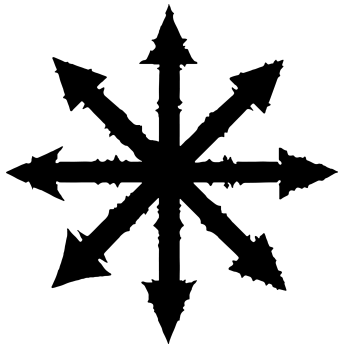
Jann Moon

Enumerating Esoteric Attack Surfaces

For Bug Bounty Hunters, Penetration Testers, Private Investigators and OSINT Aggressors

By: Jann Moon

Chapters



Preparing for Battle

Hey there,

Thanks for checking out my book on expanding esoteric attack surfaces for penetration testers, bug bounty hunters and OSINT aggressors. None of the content in this book should be too punishing in terms of difficulty, even if you are not the most seasoned gladiator of the cyber realm. However, for this tome to serve as your faithful companion in any effective manner, you should know some basic things before starting or you'll have a less than ideal experience.

It is highly recommended to use a Linux system to make this whole experience a lot more palatable for yourself. I will tolerate your set-up as a virtual machine or as a native OS but ideally, I strongly recommend that you use a cloud (VPS) box that you connect to through SSH and experience without a GUI, existing and thriving in the cold, calculating darkness of the Linux terminal.

If you are using a Windows OS, my favorite terminal application is *ConEmu*, which even allows you to run a bash shell within your

Windows OS, set Powershell to "`Run as Administrator`" by default, utilize flexibly-divisive window multiplexing, custom color schemes and the extensive array of features you'd come to expect from Linux tools like Terminator. If you do use a cloud box, you can use any provider, such as *AWS, Azure, Digital Ocean, Google Cloud Platform (GCP)* and more, all with their own neat deals for new users. After my fairly extensive personal trials with a bunch of these providers, I am currently using *Digital Ocean*.

- *AWS* slammed me with massive bills out of nowhere (Ok, they probably weren't "out of nowhere" if I had read the agreement and billing stuff more clearly) after I stopped using it. Either someone hacked into it or some other nefarious event occurred (maybe I forgot to turn it off, even), but I was really only left with one option - act confused and frustrated to customer service and whine to my bank until the charges were removed.
- *Google's GCP* was fun but at multiple points would freeze my account for up to 72 hours to do a "malware investigation". Each time I would submit a report saying that "Yes, I am hacking the Department of Defense, but it's totally fine, I am a professional and a genius" and they would give me the account back, but it was a little annoying and would hinder me from getting things done for a day or three.
- *Microsoft's Azure* has a UI design that is comparable to how I'd envision a labyrinth architected by a school of blind

infants would look like, allowing everyone using it to experience the confusion and darkness the infants I mentioned would have been born into. To its credit, the way it is organized is thematically on-brand for Microsoft. If you're working through this, I can vouch for the cloud boxes they claim to have, as I did find them eventually, so don't give up, you may even find them soon.

- Vultr doesn't allow any bug bounties or pentests regardless of permission or anything else and will ban you immediately and delete all your data without warning. Vultr takes a massive L for this.

Reasons to use a Cloud Compute Box:

- First, you are given a publicly accessible IP address that can do things like host payloads (PostMessage XSS, CORS), run servers, check vulnerabilities like blind XSS, SSRF, RCE and more. In case you are running a sensitive OSINT investigation, your personal IP will not be exposed throughout active recon measures. You could port-forward your home router and all that, but this is way easier, faster and safer.
- The internet on these boxes is a hundred times faster than my home network and can be utilized non-stop, punishing victim servers with 10,000 requests a second via FFUF. Then when they ban you, it won't affect your local IP that is running Burp Suite without all the insane scanning and server abuse tied to it. You

can often flip the IP to a new one easily and circumvent any bans that your targets try to catch you with.

- The amount of bandwidth you can blast at your targets from your home PC can't even compare to these cloud boxes. Even on the cheapest plan with any provider, I could blatantly abuse tons of targets with senseless nuclei scans, brute force endpoints, assault servers with port scans and fingerprint them continuously and still never use more than 6% of my monthly allotted bandwidth.
- Your cloud box is easily accessible from any computer, granting you access to all your notes, output from tools, firing off new scans and more. You can even check it from your phone, and yes, even while driving!! Do you think the top hunters have time to solely focus on driving at any point in the day? Highly doubtful. If you want to be the best, start valuing performance above your personal safety.
- Getting comfortable with cloud services is definitely a skill relevant to most IT jobs. Plus, setting up various tools and services can expose you to things like port forwarding, configuring servers and further advance your Linux terminal knowledge.

Usually, lots of these cloud providers offer you a fat stack of cash to incentivize you to forfeit your payment information, so be careful about running up resources past the cash bonus date. For example, they might wave 300$ in your face, but it will vanish after 30 days, regardless of how much or how little you use. Unlike a couple powerhouse service providers I've had business dealings with in the past, Digital Ocean has never launched a multi-hundred dollar bill out of the darkness at me

that left me confused, frustrated and firmly cornered me into making the only reasonable choice - calling my bank and saying that I lost my card.

If you'd like to try out Digital Ocean with a free 200$ credit and help me out a little bit, here is my referral link:
https://m.do.co/c/d129eb1d821b
Thank you! If you need any help setting up SSH, adding and mounting additional memory after spinning up your box or figuring out any of the nuances interwoven throughout this provider's available services, please don't hesitate to ask (preferably your cloud provider but I guess you can ask me too).

Required and Recommended Prerequisites

It is highly advisable that you use **Linux** for your engagements. **Debian** and **Ubuntu** are dependable, though you can bypass manually exhuming many popular tools from Github by using an offensive cybersecurity focused distro like **Kali Linux** or **Parrot (Security Edition)**. Pentesting distros like **Blackarch** and **Athena OS** are very neat, but you must accept your fate as an unchained soul, thrashing alone in the bottom of the ocean and any issues you come across will probably need to be fixed by yourself, the free-spirited commander that decided they were clever enough to end up at the same finish line as everyone else, though with many more fancy tools, handsome Desktop environment and whatever else you gain from taking this lonely, hostile road.

You should *totally* set these up in a cloud environment, however, I can't stop you if you decide to use a virtual machine, Vagrant, dual-boot OS, Docker or whatever workaround you devised, thinking it would be better, quicker, cheaper or easier. The only one that could be somewhat reasonably argued is "cheaper", but that is only if the value that you place on your time is significantly lower than you should, for the sake of your own self-esteem.

I will assume you know how to maneuver through your Linux environment via Terminal and feel comfortable performing common user actions like moving throughout your file system, changing file permissions, installing tools through Github, Python and Golang and common actions necessary to use Linux as an advanced user, a bar that appears high to the general public but anyone could get there in a couple weeks with average effort. Basic Bash scripting is not required but will definitely make your life easier through being able to automate things and not having to remember every script and argument for the many tools you will use throughout your time as a vulnerability researcher and hunter. Also, for most of the tools, I am saving the data to a file by piping the output by appending **" | tee -a tool-domain-output.txt".** If you want to be very powerful and super smart, you can use dark genius Hahwul's **gee** (https://github.com/hahwul/gee), which is **tee**, but with many added features like uncoloring text, filtering via built-in grep, splitting strings, converting the output to JSON, Markdown or HTML and more too.

Finally, in my Linux host, I always run as the “root” user. Since I am way too busy to type “**sudo**” all the time (even with my alias of “**s**”) and I am way too smart to break anything by accident, I recommend you become confident, fearless and value your time and become the actual owner of the piece of technology that YOU bought. Don’t let anyone tell you how to run your Linux sessions, unless they are offering you absolute power without regret or restraint (like I am, right now). So you won’t see much of “**sudo**” in this book, because I am a super user from the second I touch the keyboard and you should become a leader, by following this advice.

Note: *At the time of this writing (January 2024), the current version of Python is 3.12. There is a surprisingly high amount of scripts that are broken and won’t run with this version, so I generally install and run Python tools with* `***python3.11***` *to avoid this error. You can try with 3.12 but if the tool is vomiting error messages across your terminal, this is the first thing I would try to do to fix it. This includes running setup via* `***python3.11 setup.py install***` *and* `***python3.11 -m pip install -r requirements.txt***`.

Scopes and Strategies

Pioneers in subjugating classification systems for engagement variables have generally split scopes into three main categories. Split into loosely-defined groups based on the number of assets included, how many the client or program manager considers as worthy of securing and other factors, it is actually very simple to parse the campaigns, engagements and programs into these three defined categories. I first heard of the term "*Scope Based Recon*" by Harsh Bothra (harshbothra.tech) in an article he authored, defining differences in methodology dependent on how the target defines scope for the engagement.

While I think ***everything*** should be in scope, as it would benefit the hunters, as well as the hunted, through the discovery of more weaknesses within the attack surface they are aware of, as well as the discovery of assets within their attack surface that they are not aware of. In my opinion, the strongest argument for this position is the fact that any malicious actor will not care about any defined scope, potentially using lateral movement and privilege escalation to dance in and out of

any imaginary boundaries being set for the people you hire to do “attack simulation”. Yet each self-imposed limit punches holes into the realism and by transitive property, value, in accurately evaluating the security readiness against mean and clever bastards in the wild. Throughout social engineering, physical attacks and supply chain exploitation attacks that permeate the threat landscape, it's clear that there’s plenty of attack surface for an organization to try to secure, without it imposing further restrictions on hunters and penetration testers.

Anyways, I will go through this book assuming the scope is as wide as possible to show you how to uncover the most shrouded secrets, lying dormant beneath the frostbitten dirt at the bottom of a cavern hewn by time and blanketed with dense fog. When the fools that tighten the scopes on their programs, they also tighten the metaphorical noose around their metaphorical neck, while a malicious hacker metaphorically portrays the hangman, waiting to kick the metaphorical chair from under their feet, sending them into metaphorical death spiral.

Very liberal scopes give the hunter the ability to perform “*Horizontal Domain Enumeration*”, instead of the vertical kind that is synonymous with wildcard domain scopes.

- **Vertical Domain Scopes**: Includes subdomains of a target and will ideally be clearly defined as a “wildcard scope”, with a syntax that looks like “*.domain.com”, or occasionally “domain.com”. The latter, if the program owner is either a moron or prefers being

intentionally vague and wasting the time of their devs, triagers and us.

- **Horizontal Domain Scopes**: much more rare to find as it somewhat tears the cap off of the total amount they'll be paying out to hunters, as well as the total amount of time will be piled onto their developers in the form of an eternally growing list of vulnerabilities. This scope will include other domains owned by the target organization. For example, if you were hunting under a Colgate bounty program and they allowed this, you could also hunt on the Tom's of Maine toothpaste brand domains (sorry, not actually a small family-owned brand). These loosely defined scopes will often include various TLD's across their main domains and whatever other assets you can exhume from the cobwebs of the internet (though it is sometimes a bit of a challenge to prove that your target is the real owner of the assets).

If you are hunting on a target with a really impressive, loose, broad and inclusive mega-blanket style scope, discovering esoteric resources that are ripe with vulnerabilities ready to be harvested will be far more likely. With an extra juicy, thick and prominent scope, where all assets tied to an organization, umbrella, family or trademark; certain recon techniques can be used to find esoteric and neglected assets usually less hardened than the proprietary prizewinners and the triple-reinforced front entrance to the stronghold. These techniques are...

- Horizontal Domain Enumeration
- CIDR Enumeration

- ASN and IP Enumeration
- Acquisitions
- And any other recon methods available under small and medium scopes

It is very important for me to mention that recon is not a linear process. If you are familiar with OSINT, it is a similar workflow, where some piece of data will appear and allow you to discover an even greater attack surface, by utilizing steps you've already performed, but now including what you just found. For example, you can find some subdomain in a Javascript file deep into your research, which could be linked to a very different IP, leading you to discover more VHOSTs and whatever else spirals out of it. I tried my best to make the order of sections in this book logical, but there are things that connect all over the place. Like a *web*. Not so unlike the *web* you will use to find, traverse and incinerate these targets like a deadly spider, except for the part about burning your home down.

Continuous Recon

Once you enumerate the various assets (IP ranges, subdomains, JS files, subsidiaries, etc..), a crucial technique to remain two-shakes of a lamb's tail ahead of your competition is to utilize your enumeration techniques over time, comparing your new data to discover changes and prioritize

attacking them. Ideally, this should be done through automation wherever possible (and on a cloud box, with a cron job executing whenever you want, as it never sleeps) to allow you to focus on tasks that require manual parsing or doing whatever you want, as long as it doesn't include repetitive tasks that could be automated. Once your tool or script is finished, it would ideally compare the new findings to the old results and alert you of any differences.

The legends at the Project Discovery team, responsible for a very prominent slab of many web hunter's toolkits, have a flexible and dependable tool that makes notifications easy to add onto any script or tool chain.

Notify: https://github.com/projectdiscovery/notify

Install through Golang via:

```
go install -v
github.com/projectdiscovery/notify/cmd/notify@latest
```

You must fill out the configuration file for the tool to work. If you install it the way above, you can find it in your home folder at **"$HOME/.config/notify/provider-config.yaml".** If you downloaded the executable from the "Releases" section on Github, you need to download or copy-n-paste the contents of the file and put it in the expected folder. You can be notified through Discord, Slack, Telegram, SMTP, Googlechat, Microsoft Teams or any custom webhook you prefer. As you can imagine, there are all kinds of ways to alert yourself as soon as your framework discovers a vulnerability, when your blind XSS server

gets a succulent request or when that person you like leaves your message on “read”.
To pipe your tool output into Notify, that’s exactly what you do.

```
recondescending -domain babyfat.org | notify
```

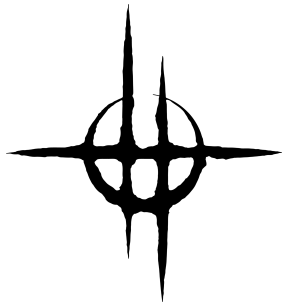
To save your output to a file, then have *notify* query it, you can run the below.

```
recondoleeza_rice -domain onlyfans.com -output plzlisten.txt
; notify -data ./plzlisten.txt
```

Personally, I like to be a little indecisive and do some of both.

```
recontroversial -domain slimes.jp | tee slime.txt | notify
```

When the scope says something very brief like “*All assets owned by [ORGANIZATION]*” for their scope, I get very excited. It produced a strong feeling that tells me I should put on some sunglasses and start kicking out the windows in my home, because I’m about to find some bounties and upgrade them anyways. With enough initiative and courage, this sentiment will resonate with you as well.



Acquisitions

Let's punch it into full actionable and useful information mode here, because most of us know what acquisitions are (it is when a company buys another company and then it falls into its cavernous wide scope). Sometimes, companies include some whiny disclaimer in their policy that says you can't test a vulnerability for the first 90 days that the acquisition becomes public (if you think you might cut ahead of the curve, explore the new acquisition before the probation period, maybe you *accidentally* send a few curious requests around and wait to report them, I'd probably advise you to check out the plethora of other programs, as you don't want to tip them off early and have them catch it in their logs and fix it).

So, where do we find out about *new* acquisitions so we can slide in before anyone else does to fill our basket with low hanging fruits?

- **Company blogs, websites and social media pages**. The official sources, so you can pretty confidently attest to their accuracy. Different organizations may be more active on one platform than

the others, so check their LinkedIn, Twitter, Facebook and Pornhub accounts to make sure nothing slips through your fingers.

- **Financial Reports and SEC Filing data.** Once an acquisition is complete, companies need to disclose it in their yearly financial reports and *Securities and Exchange Commission* (SEC) filings. Much of the data within the report won't be too relevant, beyond the names of the new subsidiaries, but you never know. Following the news of your target can also lead you to unexpected deductive reasoning patterns that may guide you towards making educated guesses about where the weaknesses are hiding.

- **Private reports published by your target.** If you purchase a stock of your company, which may only be a few dollars, they may send you various publications throughout the year, ask you to vote on tentative board members, announce plans and goals for the future and other interesting pieces of information, including acquisitions.

- **Newsletters**. Sign up for these when you see them and link them to an email address used for OSINT and recon. These newsletters sometimes send unique endpoints to surveys or whatever that can't be found through spidering or search engines.
- **Miscellaneous sources.** including press releases, industry analyst reports, stock market announcements, job postings or anything

else offbeat and interesting you come across in your hunting sessions.

Recent Mergers & Acquisitions Resources:

I am doing some groundbreaking recon research here for all my good boys and girl baddies reading this text right here. I have never seen most of these sources mentioned in any other article, book, presentation or jail kite, so if you're catching this quick enough, y'all have elevated yourselves in bounty lore to my level. Welcome to frontier-status bleeding-edge merger awareness.

- https://intellizence.com/insights/merger-and-acquisition/largest-merger-acquisition-deals/
- https://www.owler.com
- https://www.crunchbase.com/discover/acquisitions
- https://github.com/themarkib/google-acquisitions
- https://news.mergerlinks.com/

This next one is succulent but will likely dry up before the 2nd edition of this book. It is a paid source of data but can be circumvented by tricksters as clever as ourselves. There is a seven day trial you can sign up for but we need access beyond next weekend and don't want to surrender our email address as we are too busy for the account verification process.

The domain with the goods: https://mergr.com

To bypass the membership paywall, go to Google and search for:

`site:mergr.com “targetorganization.com”`

While it won’t allow you to see the full list containing the latest chronologically listed mergers and acquisitions, I searched for the most recent acquisition and it did appear, so we know that Google is quick to parse each update into its search results.

Finally, I got this method from the very skilled Jason Haddix (@Jhaddix). OCCRP follows 414 million public entities and they parse and allow you to access various datasets involving these entities. To get to work, visit the following URL, register and search for your organization’s name on the front page.

https://aleph.occrp.org

You’ll likely get a bunch of hits, some unrelated, but look out for a dataset called “US SEC (CorpWatch)” and click on the company name (not “US SEC (CorpWatch)”. Head over to the “Assets and Shares” tab and fatten your dataset.

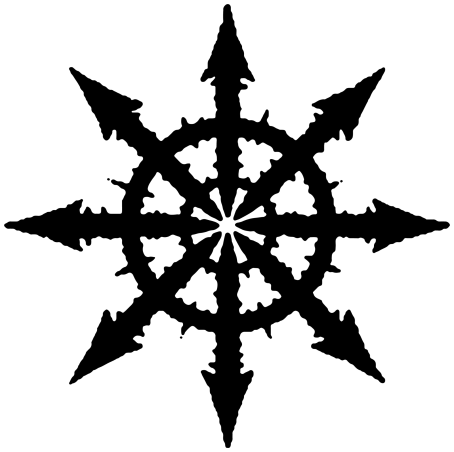
Zseano Accidentally Leaks Power Level by Projecting a Beam of Exploitational Empathy

Wrapped in Deductive Reasoning into the Minds Of Developers; Reveals Delicate Vulnerability Resulting in Affixing Personal Profile to Target's Unlimited Money Faucet

Highly revered web-hacking shotcaller *Zseano* (creator of bugbountyhunter.com, "*Zseano*" at HackerOne, X, Medium and more) described following a target's financial news and how it helped narrow his focus onto the most likely vulnerable-prone elements of their attack surface.

Through his recon, *Zseano* estimated that the target was struggling to boost their financial standing as they edged closer to the date where their quarterly report was due to be released. Underperforming financial reports send a strong signal to insecure stockholders and they begin to psychologically torment themselves until they are actively deciding to cut their losses, shoveling their money bank into the banks and hedge funds. A mastermind like *Zseano* can look at a situation like this and translate his findings into something valuable and actionable. Knowing that the target organization would be desperate for a boost in revenue as soon as possible, certain safety procedures in the software development lifecycle would descend in importance, as pushing the new features that could generate revenue to the clients would be the top priority.

He pulled the pages from the target and compared the code base to prior data, discovering the hastily implemented features and getting to testing. The features were dropped to generate revenue as he predicted and he started testing the payment feature with “dev-mode” test cards (for example, 4111 1111 1111 1111), experimenting with different countries and banks, until discovering the right combo and accessing an alternate payment system. This system did accept the test card and he was able to run as many ads as he wanted to for free on the testing domain. The lesson here is that a deep and thorough mindset for recon, coupled with continuous discovery, can lead you to an unexpected vulnerability. Once he cleverly positioned himself towards attack surfaces likely to crumble, the path towards exploitation could be followed logically for a hunter with a solid understanding of foundational hacking techniques.



ASN (Autonomous Service Numbers)

Starting our recon with a wide net and then falling into more obscure and esoteric crevices to find the hidden treasures seems like a logical approach, so ASN enumeration seems like a logical place to start. If your target has a powerful enough internet presence that they register blocks of IP numbers to host their many domains and web services and they host the content on their own servers (rather than using a CDN like Cloudflare or cloud hosting, like AWS, Azure or GCP), they likely have registered their own ASN. You should always check the owner of the ASN or the IP ranges within it, if it belongs to a cloud or CDN provider, there will be no point scanning the IPs within it and it may even be illegal. Owning your own servers and ASNs has become less common since cloud providers have become commonplace, however, if you find an ASN and verify that your target is responsible for the number and its subsequent IP ranges, all is good.This can be very helpful to us, because we can set our sights on a block of IP addresses and comfortably perform active recon on them, as we know they are definitely owned by the organization.

ASNs exist so that **BGP** (Border Gateway Protocol) can identify how data travels throughout the internet. ASNs benefit organizations by allowing them to define and implement their own routing policies. This means they can control how data flows in and out of their networks, optimize various performance features, implement security measures or otherwise manage traffic. Depending on geographical region, ASNs are managed by regional internet registries (**RIRs**) like **ARIN** (American Registry for Internet Numbers), **RIPE NCC** (Réseaux IP Européens Network Coordination Centre), **APNIC** (Asia-Pacific Network Information Centre). It is also important to note that an organization may possess more than one ASN, if they want to connect to multiple ISPs, have backups or any other esoteric reason.

Sometimes you may be lucky and have the ASN defined for you in your scope from the organization itself but other times you may need to do a tiny bit of work (really, it really is not much). Once you go straight for the ASN and begin parsing the organization's assets, you can find direct IPs that allow you to bypass WAFs, plenty of leaky SSL certs and various hidden hosts and services.

You can grab the ASN from searching for the organization or the ASN on https://bgp.he.net.

You can also query whois data, which is a database of domain owners, their addresses, their e-mail contact and their names. You can do this via querying **whois** via the terminal, as seen below.

whois -h http://whois.cymru.com $(dig +short http://target.com)

Here's some ways to punch an organization right in the ASN:

I. Using **Amass**. (Download the binary at https://github.com/owasp-amass/amass or via `**sudo apt install amass**` in Kali, Parrot, Blackarch, Homebrew, Winget and Void Linux)

amass intel -org <ORGANIZATION-NAME>

amass intel -asn

ii. Query **ipinfo.io** and ASN will be given in the response.

curl http://ipinfo.io <ip>

iii. Using Harleo's excellent tool **Asnip**

go install github.com/harleo/asnip@latest

asnip -t <IP or Domain> -p

iv. Using nitefood's very useful tool **asn.** Though the other tools are concise and follow Linux's "do one thing and do it well" credence, this tool overwhelmingly makes up for the void of creativity present when it came time to name it. Beyond just ASN, the tool delivers organization

name, RIR region, IXP presence (geolocation of the server), global AS rank (based on multiple factors) and a bunch more. It queries a lot of free and paid API services, including Shodan, Greynoise, Whois and more. Since it offers quite a bit of functionality, there are some requirements beforehand (assuming you are using a Linux box like Ubuntu, Debian, Kali or Parrot) and then the installation script (no malware, I promise).

```
apt -y install curl whois bind9-host mtr-tiny jq ipcalc
grepcidr nmap ncat aha
```

```
curl "https://raw.githubusercontent.com/nitefood/asn/master/asn"
> /usr/bin/asn && chmod 0755 /usr/bin/asn
```

Usage follows a simple syntax, and its usefulness might earn it justification for the name "**asn**".
For target, you can use ASN (both AS999 and 999 formats are acceptable, case-insensitive), IPv4, IPv6, organization name, domain or URL (even with wonky ports and protocols)

```
asn [target]
```

iii. Using ProjectDiscovery's **asnmap**, which matches the quality of their other tools, spoiling us these past few years. You can install it by building it yourself with Golang or downloading the executable in the

format of your choice here:
https://github.com/projectdiscovery/asnmap/releases

go install github.com/projectdiscvoery/asnmap/cmd/asnmap@latest

Usage also allows targeting a versatile bunch of formats :

asnmap -a ASN

asnmap -i IP

asnmap -d DOMAIN

asnmap -org ORGANIZATION_NAME

asnmap -f FILE_CONTAINING_TARGETS.txt

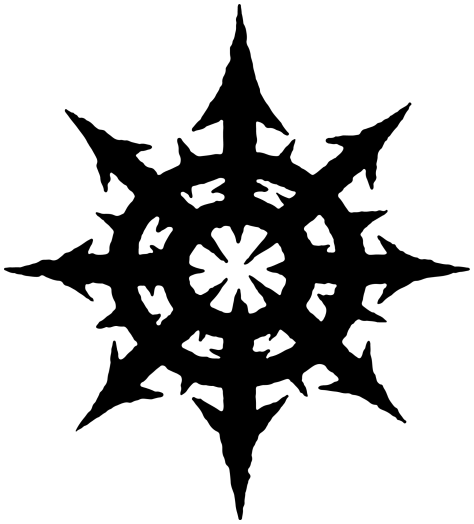
Once you have the ASN, you can then search for it on Shodan, Censys, Fofa, Zoomeye and other similar services (more on those later). Below is the syntax shown through some example queries.

- Shodan dork: **asn:as394161**
- Censys: **autonomous_system.asn:13335**
  - **autonomous_system.name:"CLOUDFLARENET"**
- Zoomeye: **asn:42893**
- Fofa: **asn="8094"**

If absolutely everything is falling apart, here are two more places to get ASN and IP range information.

https://asnlookup.com

https://ipv4info.com



Reverse Whois

When registering domain names, the information is added to a centralizing and publicly accessible database known as the Whois repository. It keeps track of registrar names, e-mail addresses, associated organizations, domain owner, domain registrar, registration and expiration dates, and domain name servers. You can use this to your advantage by finding out aforementioned information on your target and then searching the same value across the entire database to discover other domains owned by your target.

For a browser-based way to pull some data from this method, there's a few options.

Reversewhois.io is free but limits output per query to 1,000 results, which is sometimes less than 10% of what we can get, depending on our target. Also, there isn't a super clean way to get the output into a useful format but I generally copy and paste the list it returns and save it to a file, so that it looks like this:

```
1  1800rentcat  1999-03    MARKMONITOR
       .com        -15          INC.

2   321cat.com  2007-09    MARKMONITOR
                   -22          INC.

3  800caterpil  2002-02    MARKMONITOR
     lar.com       -13          INC.

                            [snip…]
```

Then pull the second column only with choose (or awk if you hate yourself).

cat reversewhois-output.txt | choose 1 | tee cleanoutput.txt

A paid option that is well worth the investment is https://www.whoxy.com/reverse-whois/. You get 1,000 lookups for just 10$, which I bought a few years ago and is finally coming close to running out after a fairly heavy battery session. There's also multiple tools that utilize this API key so it can easily be integrated into your own automation script or the tools you're already using.
One such solution is Gwen001's **related-domains** tool, which you can install via the following:

git clone https://github.com/gwen001/related-domains

python3.11 -m pip install -r requirements.txt

Run the tool with the **-k** flag for API key and **-d** for target domain.

python3.11 related-domains.py -d example.com -k APIKEYGOESHERE

You can also query the tool directly using the syntax below. Options for the mode parameter are domains, email, organization.

curl https://api.whoxy.com/?key=xxxxx&reverse=whois&keyword=babyfat&mode=domains

CIDRs and IPs

Assuming you have your target's valid ASN, you can pull the IP ranges under it via the search function on https://bgp.he.net or through the mxtoolbox site here: https://mxtoolbox.com/SuperTool.aspx
Choose "ASN Lookup", type in your target's ASN via the format "AS9999" and you will be gifted the IP or CIDR ranges.

Using asnmap to get CIDRs from ASN or otherwise.

asnmap -silent -a AS394161

asnmap -silent -org GOOGLE

asnmap -silent -d domain.com

There is also a barebones command you can fire off to get your ASN and its associated IP ranges. [1]

```
$ whois -h whois.radb.net -- '-i origin AS36459' | grep -Eo "([0-9.]+){4}/[0-9]+" | sort -u
```

An alternative way to find these ranges that will most likely always be available, is to use the hacker security blanket known as **nmap**. This tool will be explained in further depth in the "*Ports"* and *"Fingerprinting"* sections of this sacred tome, since nmap's extensible and adaptable scripting language capabilities are capable of wearing quite a few hats. There is an Nmap script to find IP ranges that belong to an ASN that https://nmap.org/nsedoc/scripts/targets-asn.html

```
nmap --script targets-asn --script-args targets-asn.asn=17012 > paypal.txt
```

Clean up the output from the above nmap result, take all the IPs in a file and then run version scanning on them or masscan on them.

```
nmap -p- -sV -iL paypal.txt -oX paypal.xml
```

[1] Barebones, as in you could close your eyes and point at any random, hairless, newborn Linux distro and you should still be able to pull off the command, because "**whois**", "**grep"** and "**uniq**" are (or at least, should be) standard to all of them. If you ever use an unfamiliar Linux distro and one of those tools responds with an error, you should use your OSINT skills to find the developer and humiliate him, in the same way that he shamed us both when he made the choices he made.

Once you have a list of CIDRs, you want to turn them into a list of IP addresses that are hosting servers or services that you can investigate and destroy. I like to use a tool called **prips**, to expand CIDRs into a list of all IP addresses belonging to it.

If running a Linux distribution like Ubuntu, Debian, Kali or Parrot, you can install it from **apt**.

```
sudo apt install prips
```

Otherwise, you can get this version from Go, that is not the exact same tool, but it does the same thing.

```
go install github.com/imusabkhan/prips@latest
```

To use it, follow the syntax below.

```
prips 10.10.10.0/24
```

Save this list as it is, for fuzzing VHOSTs later on. To do something interesting with this list right now, we can feed them into a tool that does reverse DNS lookups. As DNS servers normally work by feeding them a hostname (which is easier for users to remember) and the DNS server will return the IP address, so your browser can find the content it

wants. *Reverse DNS* works in much the same and opposite way, I think you get it.

A no-nonsense or funny business tool that does this, is charismatic Hakluke's tool, **hakrevdns**, available at https://github.com/hakluke/hakrevdns

You can funnel your IP straight from **prips**, as seen below.

```
prips 10.10.10.0/24 | hakrevdns | tee hakrevdns-CIDR.txt
```

You can also set specific DNS resolvers for hakrevdns, which will produce some subtle differences in results. Currently, the best and most updated lists can be found here:

https://github.com/trickest/resolvers

After running **hakrevdns**, you may notice that the output may not be ideal for piping to another tool. You will also probably want a list of just IP addresses that hit hosts and there are a couple ways to do this, but one will accelerate your race to the arthritis finish line at a significantly faster rate.

Just for historical accuracy, here is the way that makes your bones tingle.

```
cat hakrevdns-CIDR.txt | awk -F "\t" '{print $1}'
```

Output:

165.26.210.227

165.26.210.229

165.26.210.253

165.26.211.3 *(and so on…)*

OK, some of you will probably say that **awk** has a lot of other uses and it is very versatile but I've been lucky enough to live during an era where I can find other tools to fulfill those uses and avoid the deep dive into **awk** that many Linux users likely drowned in. Educated hunters of the modern age will be pleased to discover an alternative tool called "**choose**" can produce an identical output at a faster speed, available as a pre-compiled binary for Linux at https://github.com/theryangeary/choose/releases/
The charming syntax can be seen in action below:

```
cat hakrevdns-CIDR.txt | choose 0
```

Choose can predict how you are trying to parse the data that you feed it and the number corresponds with which section you want it to output. The count begins at 0, like almost everything else in programming except for **awk**. To grab the last field, use "**choose -1**" (and "**choose -2**" for the second-to-last field and so on). You can also signify which field separator to use with the "**-f [field_seperator]**" syntax or output multiple fields like below.

```
cat data.csv | choose -f "," 0 3 5
```

Slapping around IPs until they leak juicy tidbits must be a specialty for Hakluke, because he has yet another method of expanding attack

surface from a list of IPs again. Hakluke's **hakip2host** acts similarly to his reverse DNS tool, but it queries DNS PTR lookups, as well as Subject Alternative Names (SANs) and common names (CNs) on SSL certificates. As certificates have their own section a bit later in the book, I will let myself tell you about them a little later. For now, just think of them as the tickets at Chuck E. Cheese, except for some worthless plastic vampire teeth that have never been worn on any Halloween, they expand your target's attack surface, which can be worth thousands of dollars.

Install the tool via go:

go install github.com/hakluke/hakip2host@latest

And use it much the same as his other tools that punch IP addresses.

prips 173.0.84.0/24 | hakip2host

Output:

DNS-PTR] 173.0.84.23 new-creditcenter.paypal.com.
[DNS-PTR] 173.0.84.11 slc-a-origin-www-1.paypal.com.
[SSL-CN] 173.0.84.67 api.paypal.com
[SSL-SAN] 173.0.84.76 svcs.paypal.com

If you wanted to create a file with just the subdomains, you could do so like this.

prips 173.0.84.0/24 | hakip2host | choose -1 | sort -u | tee hakip2host-subs

Or just the IP addresses

prips 173.0.84.0/24 | hakip2host | choose -2 | tee valid-IPs.txt

Or if you don't care about where the data came from.

prips 173.0.84.0/24 | hakip2host | choose 1 2

Let's push our recon further to try to bypass reverse proxies that may be protecting the origin IP of a domain. Allowing you to target the content directly through the IP in your browser, bypasses WAFs, load balancers and some other protections that your target organization has outsourced to third-party vendors to protect their assets. While you can report the origin IP discovery as a vulnerability, it is somewhat low in severity, and the organization's goons will likely say something smug and predictable like "*So what if you can get the origin IP? It is so secure that we don't even need the WAFs protection, unless you can prove otherwise?"*, so it is ideal to find another vulnerability like an XSS or SSRF to chain it to. Anyways, Hakluke has a tool for this purpose called **hakoriginfinder**. It works by sending a request to the IP address and the target host, then comparing the response for similarity using the Levenshtein algorithm.

Install it via:

go install github.com/hakluke/hakoriginfinder@latest

Then serve up **prips** with your CIDR range and set the **-h** flag with the host you want to test for with **hakoriginfinder**.

prips 93.184.216.0/24 | hakoriginfinder -h https://example.com:443/foo

You can also parse the data to ignore all the "NOMATCH" results and remove the "MATCH" word on the remaining output, so you are left with the more pipe-ready "*protocol://IP_Address:Port*" syntax.

prips 93.184.216.0/24 | hakoriginfinder -h https://admin.example.com/ | grep -v "NOMATCH" | sed 's/^MATCH//g' | tee hakoriginfinder-output.txt

You can also use a tool like imAyrix's **cut-cdn**, to remove IPs that are protected by WAFs and load balancers or hosted on cloud providers. Since they host multiple unrelated domains on various ports, doing any kind of scanning will be out of scope, get you blocked by the provider and disappoint you when you think you found something overflowing with vulnerabilities, but find out that it actually belongs to a Mongolian basket-weaving forum and they are not interested in paying you for your hard work. Even with the WAF's protection, you can still look for

vulnerabilities on the domain (the one that is actually in your scope) and wrestle their defenses, hopefully leading to smuggling some nasty code past them, but any kind of attacks or direct access via the IP will be off-limits.

Install **cut-cdn** via go (no “Releases” , so time to figure this out, if you haven’t yet).

```
go install github.com/ImAyrix/cut-cdn@latest
```

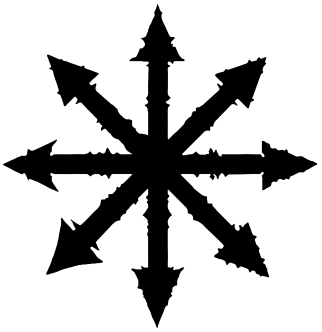
First, it's a good habit to update both the cloud providers (**-ua** flag) then the ranges (**-ur**) to have accurate data. Skipping this step and hunting on a target that isn’t in scope makes you feel like a badass when you find a vulnerability and can’t turn it in (or dumb-ass, smooth brain, whatever your inner bully tells you after you fail things).

```
cut-cdn -ua ; cut-cdn -ur
prips 10.10.10.0/24 | hakrevdns | cut-cdn
```

And here, a few web-based tools that serve up similar information, just in case the world descends into some kind of hell spiral and Github is tortured out of existence.

https://centralops.net/co/domaindossier.aspx

https://toolbox.googleapps.com/apps/dig/



Ports

Across each IP address, there exists 65,535 possible ports for services to run on. Although you can run almost any service on any port, services generally have default ports and many tools will parse them based on these expected ports. For example, web servers generally run on port 80 for HTTP and port 443 for HTTPS, while you will occasionally see servers running on 8080,8443,8001,5000 or any other port. At this stage, we need to have either a CIDR or a list of IP addresses so we can fire off requests at their ports to see if we can find a service that your target probably doesn't want wide open to the public, some outdated software that your target has neglected to update for some time or just more attack surface to wage war against in later steps.

First, since we may have a pretty chunky list of IP addresses to test and 65,535 ports for each one, we want a tool that is fast and violent without sacrificing too much accuracy (without sacrificing any accuracy, ideally). A long-time staple for penetration testers, hunters, people that like to have the ability to send 10 million packets per second (like me) and people that need to scan the entire internet within a few minutes (not really me), is a tool by robertdavidgraham called **masscan.** Now this is

an impressive tool, but it needs to be reigned in to get effective results. This became clear to me when I scanned the entire internet so fast that it didn't even find one open port.

A version is available on Github (**https://github.com/robertdavidgraham/masscan**) but it's easier to install via apt, and available in most distros, while not being very fun to compile on your own.

```
apt install masscan
```

I should also mention that the output is pretty disgusting, to the extent that I learned how to use **jq** to parse JSON, rather than have it burn my corneas ever again.

```
masscan 10.10.10.0/24 -p 1-65535 --rate 100000
--output-format json --output-filename scan-results.json
```

Armed with our list of open ports, we need to find out what is actually running on them and whether the answer to that question deserves further inquiry, probing and punching. **Nmap** has been a long staple in the hacking and infosec community and is usually what is running on the screen if you ever see someone "hacking" in a movie. With **nmap**, you are able to check for open ports on local and remote networks, check for live hosts and even do a good bit of service enumeration. Through the use of scripts, **nmap** has been extended to be able to brute force logins, perform user enumeration, check for CVEs and anything else there is a script for. Many hackers run **nmap** as their first

command to get a quick, comprehensive, surface-level understanding of a target, informing them where to focus next.

You may likely have **nmap** on your box already, but if not, then install it via:

```
sudo apt install nmap
```

Before running **nmap**, there is a worthwhile script that doesn't come bundled with the tool. To add it, download the following file, move it to your nmap scripts folder and then update your tool's script database via the commands below.

```
wget
https://raw.githubusercontent.com/vulnersCom/nmap-vulners/master/vulners.nse
```

```
mv ./vulners.nse  /usr/share/nmap/scripts/vulners.nse
```

```
nmap --script-updatedb
```

Nmap has a vast dune of possible arguments for doing all kinds of wonky things, but I'll list the ones most relevant to the type of person this book is for.

-iL : Scan targets from a file

-A : Unlike popular opinion, this “A” does not stand for “all”, but rather “aggressive”. This flag enables OS detection, version detection, script scanning, and traceroute

-sV : Attempt to enumerate services running on open ports and their versions.

-T4 : You can enter between 1 to 5 to signify the speed that nmap will fire off packets. The default is 3 and I’ll nudge it up a bit if I can get away with it, occasionally using “2” if the WAF is very finicky and abrasive to my requests.

--script [script or category] : To specify which types of scripts you want to run against your target, you can either list them individually or use preset bundles of scripts by writing “**auth**”, “**vuln**”, “**intrusive**”, “**discovery**”, “**exploit**”, “fuzzer” and a few more.

You can find all their names and descriptions here:

https://nmap.org/nsedoc/categories/

Definitely use the *vulners* script mentioned during the installation of this tool. A basic and useful way to drop it on your enemies is below.

```
nmap --script nmap-vulners -A -sV 192.168.1.10 -p 1-65535
```

I mentioned earlier in the book that recon (nor any workflow in hacking) should be looked at as a linear checklist. While having a methodology, being organized and following your plan will help you significantly avoid wasting time, becoming overwhelmed, performing redundant work and forgetting to look for certain things, an attack plan that is

responsive to nuanced and subtle oddities throughout testing is best. You may start with a single domain and its corresponding IP address and set your session off with nmap, like an OG, and

Another alternative you can use, with a good amount of speed, as well as accuracy, is projectdiscovery's "**naabu**". It incorporates **nmap** but it is often wonky so I'd just use it for port scanning and then do a separate **nmap** scan that you can fine-tune the scanning and the output. Grab a compiled version from:

https://github.com/projectdiscovery/naabu/releases

Or Compile it yourself with Go.

```
go install -v
github.com/projectdiscovery/naabu/v2/cmd/naabu@latest
```

Using it is fairly simple, read the docs if you want a deep dive but I don't think anything will surprise you much.

```
naabu -host gentrifier.com -p 1-65535 -o output.txt
```

For the best overall output, my favorite tool for port scanning has been **Rustscan.** Some people say they've had issues with this tool's accuracy but I think they're haters and losers because I've never experienced this. Get the tool from the releases, because tools written in Rust are

awesome to use but compiling them is akin to being waterboarded in nail polish remover.
https://github.com/RustScan/RustScan/releases/

Assuming you're using an Ubuntu/Debian/Kali/Parrot host, install the file you grabbed above like so:

```
dpkg -i rustscan_1.10.0_amd64.deb
```

You can run **rustscan** like so:

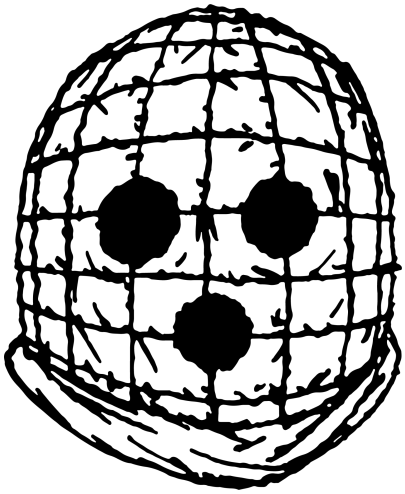
```
rustscan -a [target domain/IP/CIDR]
```

You can use additional flags to set the rate of scanning and decide which ports to scan, but the speed is very commendable and you want to scan all of the ports, which is the default. It will also automatically funnel its results into nmap with some basic values, but to set your own, use two consecutive dashes (--) and follow them with the **nmap** flags you want to use, like below.

```
rustscan -a 10.10.10.10 -- -A --script vuln -T4 -sV      -oA
nmap_results
```

After you find all the open ports for your target's hosts, you can then do some research into fingerprinting the service further (later in this book), looking for CVEs and known exploits(maybe in the next book) and

trying to figure out some clever way to bury it (don’t actually DoS it or otherwise take it offline).



Third-Party Services

Some individuals decide to make it their job to be super overbearing and nosy into everybody's business (literally and globally), poking and prodigy as much information from every server in the entire existing IP range as is legally and ethically redeemable. Ultimately, these individuals decide to start organizations and profit off of low-key spying on everybody and placing the blame on individuals who haven't had the opportunities to learn about web security or the devices they own. Anyways, it is actually pretty awesome and they have my total support, whether they are surveying the cyber-landscape or grossly invading individual's rights and profiting off of them. They usually claim they do it for an altruistic reason to increase security awareness but I wish they would be honest with me. Hey Shodan and Censys and Greynoise! I think it is genuinely awesome that you expose dummies to malicious individuals and keep the world entertaining.

Shodan

Known lovingly as the hacker's search engine, **Shodan** scans the entire range of possible IP addresses and their top ports, making the information searchable by service, product, OS, country, keyword and more. While most don't utilize it to its full potential, some have been very successful and used Shodan to discover well paying vulnerabilities (check out talks by OrwaGodfather for Shodan mastery techniques). There is benefit to using the web interface, as opposed to just the terminal package, unlike most tools. You can certainly use Shodan to search for domains, IP addresses or keywords but probably the best filters to use with Shodan are the following:

Ssl.cert.subject.CN:"overlord.com"

"ssl:Swinegames Inc."

If you get tons of results, you can filter them via negative filters by adding something like:

-http.title:"Page Not Found"

-http.status:"403"

You can find a list of all the Shodan filters here:

https://www.shodan.io/search/filters

The command line tool also has its uses and can be installed via the following command.

python3.11 -m pip install shodan

To perform a search:

shodan search "MongoDB Server Information" port:27017

shodan search "Citrix Applications:" port:1604 org:”Yahoo”

Get a list of subdomains, IP addresses and more about a domain.

shodan domain dhs.gov

Search for a specific IP address:

shodan host 104.16.149.244

One interesting way to search for new attack targets via Shodan is through Google Analytics tags. You can often find it within the content of a sites main Javascript file (typically app.js or main.js) or sometimes within the HTML code. It will look something like: `GTM-TL2KT9H` or `UA-1592615`

You can search the body of server front pages for this Google Analytics code via this filter:

http.html:'ua-1592615'

You can also use Shodan to scan its database by favicon hashes. Favicons are the tiny 16 x 16 pixel image icons that show up next to each tab in your browser, serving as an insignia to the organization or if not specifically defined, sometimes the framework that the site is running

on. Anyways, there are tools that exist that create a hash of a site's favicon and then you can search that hash value across Shodan to find other related domains and subdomains.

One such tool is **MurMurHash** by Viralmaniar. To install and run the tool, follow below.

git clone https://github.com/Viralmaniar/MurMurHash.git

python3.11 -m pip3 install -r requirements.txt

Then run the tool via:

python3.11 MurMurHash.py

The tool will ask you to enter the URL for the favicon you want (just enter the main URL, not necessarily the endpoint of the favicon.ico image file). If a valid favicon is confirmed, you'll receive a favicon hash in the form of a 10-digit number, that you will then copy into the following Shodan filter. The example below uses the favicon for Spring Boot, for which Shodan will list all of them across the internet.

http.favicon.hash:116323821

You could also use the command-line version of Shodan as well, like so:

shodan search http.favicon.hash:116323821 --fields ip_str,port --separator " " | awk '{print $1":"$2}'

If searching for a specific technology (like Ruby on Rails, Spring Boot, etc), you can add the **org:"Target Organization"** flag to the search as well, to prevent getting millions of results.

Another excellent tool that gives you some quick details about your target quickly is Shodan's own **nrich.** Since it is written in Rust, it works really well and really quickly but can also cause despondency to those who try to compile it themselves. For those who have in investment in retaining their own mental health, you can grab a release for all platforms here:

https://gitlab.com/shodan-public/nrich/-/releases

As nrich only accepts IP addresses as an input, you can resolve your domain or subdomain list into IPs with projectdiscovery's **dnsx** (described later on, in the section about resolving permuted subdomain names). The command to do this is below.

```
cat subdomains.txt | dnsx -any -resp-only | nrich -
```

Here's a few more awesome Shodan relevant resources to keep you busy.

An online interface to search Shodan through various dorks to find vulnerabilities and interestingly weak services:

https://mr-koanti.github.io/shodan

Some lists of neat and interesting queries:

https://www.osintme.com/index.php/2021/01/16/ultimate-osint-with-shodan-100-great-shodan-queries/

https://github.com/jakejarvis/awesome-shodan-queries

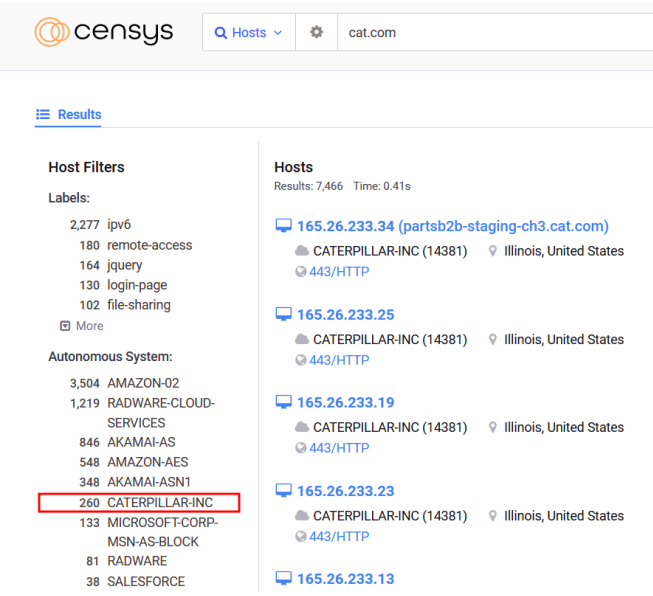
And some other good info:

https://community.turgensec.com/shodan-pentesting-guide/

CENSYS

Another similar service I need to mention is **Censys**, found at https://censys.io. Oftentimes, I find Censys to be more comprehensive than Shodan and I believe it scans more ports than Shodan does. Make sure you register an account and head to https://search.censys.io to get the most of the service. You can search for domains, IPs, ASNs, CIDR ranges or whatever else you may be looking for.

One way to use Censys is to try to expose origin IPs for servers that are protected by Cloudflare or another provider. To do this, use the search to look for your target domain. Then, look to the left column under ASNs. Try to find the ones that belong to your organization and you will be left with IPs that are generally unprotected by WAFs and are ripe for heavy port scans.



There are also other similar services that are definitely worth checking out on your own. **Zoomeye.org, quake.360.net** and **Fofa.io** are a bit tough to navigate and register for as they are in Chinese but if you can find an API key on Github, I strongly recommend using it in your tools. Services based in the greatest country in the world include **viz.greynoise.io, hunter.how, onyphe.io** and **app.netlas.io.**

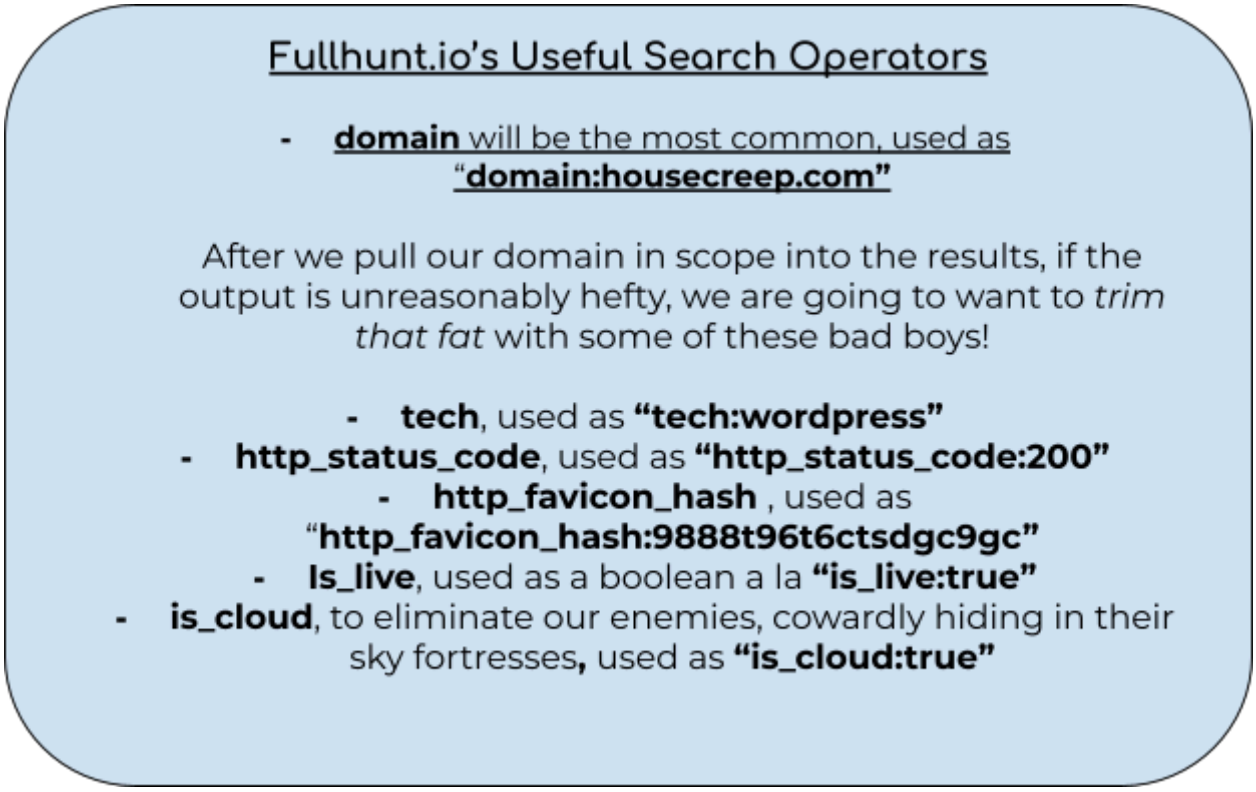
Some even list the CVEs that the server is vulnerable to, check it out on **netlas.io**.

But you already probably knew about Shodan and Censys, right? All of us have spent more nights than we'd care to admit watching middle-aged Russian women working in a Siberian post office or found some offshore vessel and pressed all the buttons we could find until some alarms started ringing. So let's dig a little deeper into some places that the normie bounty hunters won't even consider looking into.

FULLHUNT.IO

This domain is pretty awesome and even beats Shodan and Censys in presentation and depth of data, just not the breadth of data available. Signing up is quicker than it takes a cat's whisker to twitch and then we

can start piling on the queries. It matches most of the search operators available in Shodan and Censys, while simplifying the ones that Censys constructs in its own convoluted way and transforms them into the way a rational human would expect them to appear.

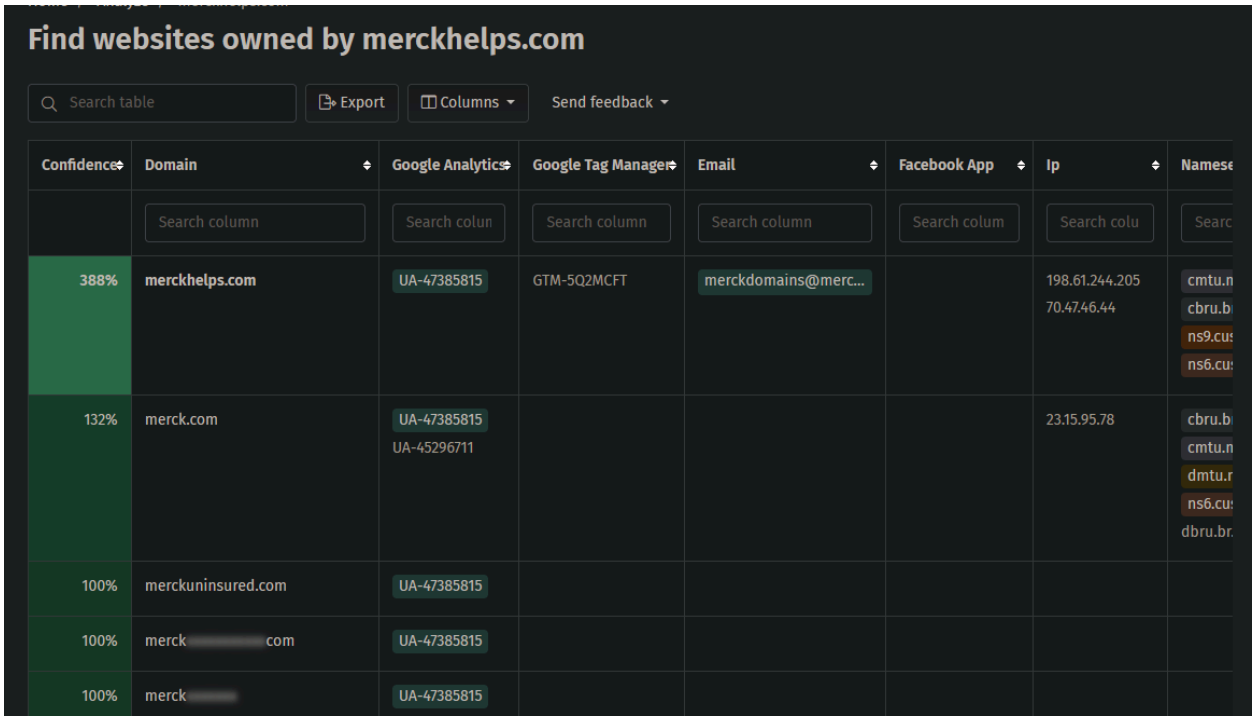


Fullhunt.io is a pretty cool guy, as it actually found more servers than Shodan or Censys and presented the information in a more aesthetic way. There was also more of it, all available to the free account that allowed me unrestrained access to the data they decided to give me access to. One place where fullhunt.io doesn't annihilate their competition is in their ability to spy on the entire internet. Multiple domains that I tried to enter had no data at all, though I'm certain that Fullhunt plans to expose and document every corner of the internet eventually and their ability to do so will coincide with their aggressive

growth investing in more probes and scanners. Anyways, this site looks HOT. Don't take my word for it, feast your eyeballs below!

Another resource that I've never seen mentioned in any hunter's methodology or resources lists is **AnalyzeId** (https://analyzeid.com) . If you thought that these guys solely uncover associated domains through reverse IPs, then you'll feel rightly humbled when you visit their front page and are specifically told that they do "*More than just Reverse IP*". If your fragile concept of consciousness hasn't been shattered, you can then use the site's functionality to try to enumerate some more assets for your target. The service is targeted towards SEO professionals and domain flippers (or upsellers? Internet branding vulture-revivalists?) so many bug hunters will be sleeping on this

service. That would be a lot more empowering if you weren't about to see the shackles they mummify you with when you search for your target domain.



If you look closely above, you can see that they grant you just two miserable results before tormenting you with the rest of your queried data, albeit smudged and out of focus conspicuously on the sections that you really want to read. I'm hoping these cruel business practices change someday, either by the warming of their hearts, by legislation or by violent force, as I don't know any other services that use Google Analytics, GTM tags and Facebook tokens to discover related hosts. I almost always end up checking those manually when I am parsing and deobfuscating the JavaScript files but this is a quick and visually clean way to get this info.

Dorks

Using Dorks for your recon is one of the easiest and most rewarding victories you can enjoy. Dorks are filters you can employ into your Google, Bing or other search engine searches to find vulnerable things that should probably not be indexed (and unprotected). You can find some shocking shit with these and there are way too many to go over so I'll drop some sweet and nasty ones and their general category of usage below.

Using a reductive technique for subdomain enumeration is sometimes pretty awesome. First, you search for your domain like this:

`site:*.ladyboys.com`

Next, if you have tons of lame results like "www.domain.com" or "blog.domain.com" or a billion forum responses at "forum.domain.com", you can remove them with the **-** operator before your filter.

`site:*.ladyboys.com -site:www.ladyboys.com`

`-site:forum.ladyboys.com`

Continue removing subdomains until you are left with something interesting. Or nothing :(

Next, you can find sensitive documents for your target organization using the following dorks (replace Neurolink with your target organization, obviously).

site:docs.google.com/spreadsheets "Neurolink"

site:groups.google.com "Neurolink"

“Confidential” Neurolink filetype:pdf

intitle:”index of” intext:”config.yml” OR intext:”database.yml” OR intext:”settings.py” “Neurolink”

intitle:”index of” intext:”database.sql” OR intext:”dump.sql” OR intext:”backup.sql” Neurolink

intitle:”index of” intext:”api-docs.json” OR intext:”swagger.json” Neurolink

intitle:”index of” intext:”dashboard.yml” OR intext:”settings.conf” OR intext:”config.properties” Neurolink

You can also use it to find self-hosted bug bounty programs that aren’t part of the majors (H1, Bugcrowd, Intigriti) and will have significantly less competition. You can find a good list here:

https://github.com/sushiwushi/bug-bounty-dorks/blob/master/dorks.txt

You can also run through some dorks with some great tools. A common issue is Google will slap you with a CAPTCHA and mess up your whole firestorm of requests against its servers but there are some workarounds that clever people have discovered. One great tool that doesn't suffer from this issue is dwisiswant0's **go-dork** at which can be installed through Go via:

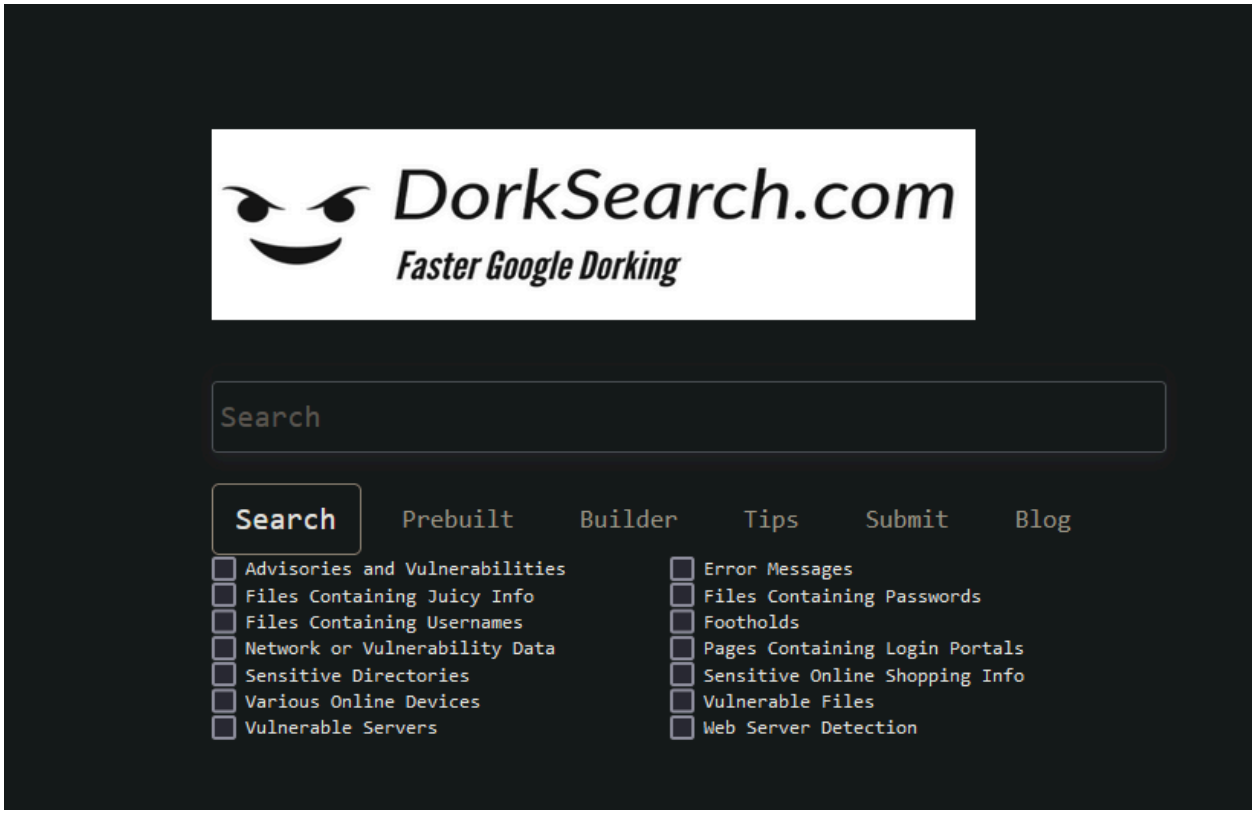
```
GO111MODULE=on go install dw1.io/go-dork@latest
```

You can run via the following syntax, with **-q** being your dork or query and **-p** being the amount of pages the tool will pull from:

```
go-dork -q "intext:'jira'" -p 5
```

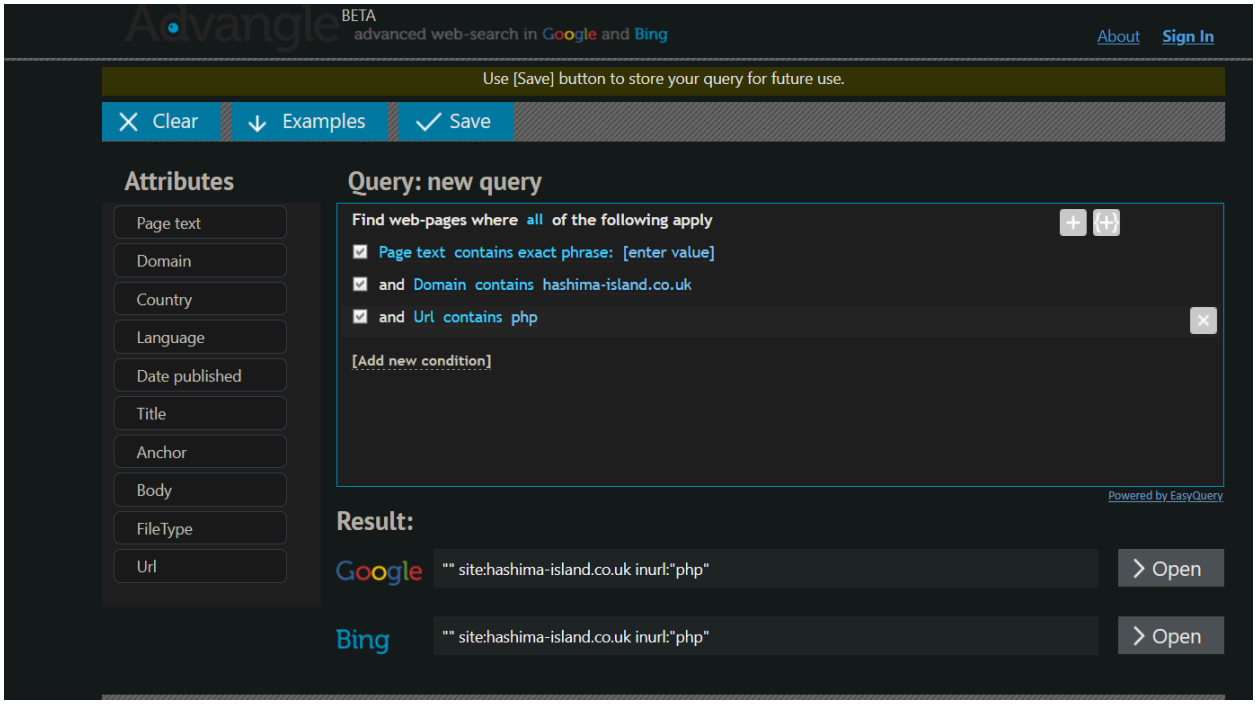
As the output is just a list of endpoints, it can be conveniently chained to other tools like hahwul's **dalfox** and projectdiscovery's **nuclei** to find vulnerabilities.

If any of this looks extremely difficult, you will still be spared, as someone has put in extra work, just so you can do far less. An entire frontend web UI has been set up to give everyone's brain a vacation and you can find it at https://dorksearch.com



As you can see in the image, you don't need to remember the operator syntax or much else, as you can pull from a list of over 7,000 pre-built queries, neatly organized into categories of variable juiciness. You can also build your own with the "builder" feature, which will hold your hand very tightly so you can't make any mistakes.

Another building service that is focused on using Google's advanced search operators, rather than abusing them is **Advangle** (https://advangle.com)



Here's a technique that may help you find the final piece of attack surface that danced through every other technique you used and remained hidden. Essentially, it is just a regex search for the domain name, though you will still need to manually verify that the asset is truly owned by your target organization. While this could be accomplished via a google dork like **"site:*target*.com"**, there is a much cleaner way that will automatically deduplicate apex domains.

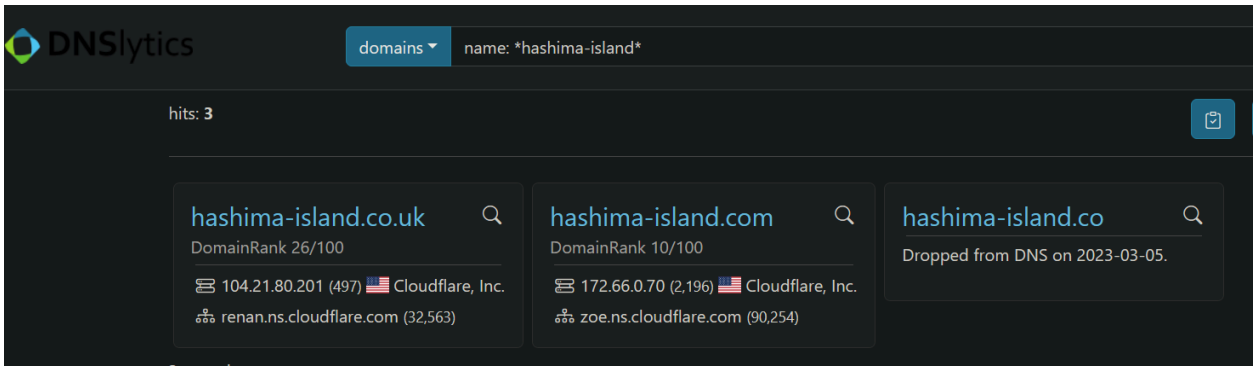
DNSlytics (https://dnslytics.com/) offers a handful of passive third-party intel services including reverse IP lookups, typo awareness (so you can find, avoid or typosquat phishing servers), parsing historic DNS records and options for monitoring domains for changes and the subsequent notifications you'd expect. Most commonly though, I use DNSlytics for a keyword-regex search across their threatening agglomeration of domains. You can access it directly by making a slight modification to the URL found below.

https://search.dnslytics.com/search?q=name:+*REPLACE_ME*&d=domains

You can also use DNSlytics "**name**" operator and utilize their "**domains**" search feature using the syntax found below.

name: *KEYWORD*

For example, if I wanted to target any sources about a small Japanese island that once had a bustling Mitsubishi coal mine, that eventually shuttered its doors in the 1970's, cleared all inhabitants from the island and became an powerful example of literal and metaphoric societal decay through its haunting remnants of brutalist architecture, I'd replace "**KEYWORD**" above with "**hasima-island".**



In the case that your target has a lengthy history (particularly of web activity), it will likely have inconsistencies between apex domains across parameters like registrars, IP spaces and many more due to the many various people making various decisions over various time periods and locations. Due to the shifting needs of an organization and the unique footprint it weaves across the web, there will never be a perfect solution to the issue of horizontal domain enumeration. This is not only an exciting and liberating challenge, but it also creates an opportunity to

excel and continue finding innovative ways to outperform your peers.

DORKS vs. GITHUB

Dorks can also be against Github to find leaky repos made by forgetful employees of the targets you are looking for. There's a lot of tools that tackle these tasks but since there are a lot of random blobs of numbers and letters through GIthub, there are a lot of false positives that show you meaningless information. In my experiments, at least, I haven't produced nearly as much interesting or relevant data as when I've done manual investigations and parsed through the code myself. Also, tools won't pick up on things that might interest you, such as secret subdomains, links to private documentation, weird syntaxes that spell out usernames or passwords and otherwise cool things that won't be caught in the filters used by the tool. Tools also can't understand context, so something that may not be sensitive via a relevant dork could still spell out directions to the treasures hidden beneath the surface. To build your own dorks, here are the most relevant Github filters:

- **filename:FILENAME** to search for specific filenames. For example, the "`config.php`" file that stores MySQL passwords for Wordpress sites. You can also do a search like "`filename:.php`" to search for all PHP files. Some other interesting ones to search for are "`passwords.txt`", "`private.key`", "`id_rsa`", "`config.yaml`", "`backup.sql`", "`dump.db`", "`debug.log`", "`error.log`", "`credit_card_numbers.csv`",

"`customer_details.xls`" or whatever neat stuff you can think of.

- **extension:EXT** will search for files that match a certain extension or file type. You can use ones like "conf" for (some) configuration files, "env" for environment settings, "pem" for files that store SSL certifications and potentially, private keys. Another useful way to use this filter is to search for data serialization format files, such as "*yaml*" (Docker, Kubernetes, Ansible, Github Actions, Jenkins), "*json*" (Terraform, AWS CloudFormation, Drupal, IBM cloud, VMWare, Chef, Puppet, many NodeJS apps and more), "*xml*" (used often by Apache Ant, Spring Framework, Tomcat, Maven, KeePass, many .NET apps) or "*properties*" (found in many Java applications like Spring Boot, Apache Commons Configuration, Kafka, Tomcat, Eclipse IDE), which commonly contain API specifications or other private information.

- **org:ORG** can be used to search for your target organization, such as `org:"Caterpillar, Inc."`

- **language:LANG** captures files of a specific language type only.

- **user:USERNAME** can let you search through all of a user's files if you happen to find some working for your target organization.

As with most things in anti-sec, having a methodology you can repeat and check off each step is conducive to finding valuable vulnerabilities or secrets. Oftentimes, it is adaptable and you change your course

based on the information you find, but it sets a good starting off point and allows you to progress to the next step instead of getting stuck and losing faith in reality. So here's a general outline of things to look for against your target.

1. Start off the recon by searching for your target domain alone **"drillerkiller.com"** (also try **sub.drillerkiller.com** and **drillerkiller**, depending on how many results you're given). A helpful filter you can use at this stage is "NOT", to filter out results that are not helpful. If you have a domain that produced a billion results, you can cull the useless ones like so:

yahoo.com NOT www.yahoo NOT blog.yahoo

yahoo.com credentials NOT www.yahoo NOT gmail.com NOT hotmail.com

Yeah, it works better when you don't add the ".com" on the NOT ones for whatever bizarre reason.

Next, search for the organization via **org:"Torture Star Video"**. If you can find your target organization on Github, you'll at least get to skip the part where you verify that the secret is definitely connected.

2. Besides repositories, you can also look at code, commits, issues, discussions, packages, marketplace, topics, wikis and users**.** Generally, I've only found useful things in the code and commits sections, the latter of which has people that things their secrets

disappear after they update their git repo after removing them from the new version.

3. If you have a ton of results that would be too ridiculous to parse through, you can chop down results via using keywords, such as **“Jenkins”**, **“oauth”**, **“JDBC”**, **“credentials”**, **“config”**, **“pass”**, **“secret”** and whatever else you think will yield some nice fruits. I’d also recommend searching for whatever language you see your target using that is not accessible, such as PHP, C#, Python, Java or Ruby. Bash can also be a good one to look for, as you may find start-up scripts or environmental secrets hidden within the code.
4. Next, you can enumerate the employees of your target organization through finding them on LinkedIn or Twitter. Oftentimes, employees don’t list their company affiliation on their Github pages but will be much more open to doing so on LinkedIn or other social networks.
5. Search in your site’s source code for terms like “**github**” and “**gitlab**” and you might find links to hidden repositories. This can be done easily within Burp Suite under the “Target” tab or if you have downloaded all the pages, you can grep through them that way.

Some additional tips:

- Always test your discovered secrets to see if they are still valid. They’re not really worth much if its an expired login, however, you could still try to brute force some common passwords under the username, discover username naming contexts or try some

altered version of the password (for example, you discovered "*Winter2019*" which no longer works, so now you can try "*Summer2024*").

- Also, be aware of the context of what you're looking at by reading a little bit of the code. If the repo is just some test data to illustrate how a tool works, it is supposed to be private.

- You can also use wildcards like "***.domain.***" in case they have a ".**net**" and a ".**com**" and whatever else for a TLD. You can do this EVEN if those other TLDs are not in scope (just make sure they at least belong to your target organization). If you find something awesome and expresses measurable business impact or an unacceptable amount of risk inherent without mitigation, then go forth and write a good report that explains these things without too much jargon. To the surprise of good samaritans (or VDP lifers), they will many times accept your findings and reward you.

- Set your search results to "Recently Indexed" to get the most recent results, for obvious reasons.

- Github leaks are very time-sensitive, as there is always a race for the credentials between bug hunters, the individual that made the mistake, the target organization's security team and actually malicious entities. A top recon tip is to save the Github search results into a folder of bookmarks in your browser, then you can rapidfire click your bookmarks and see if anything interesting has

appeared. There's totally a way to automate this and make the requests without opening the browser yourself and playing "Spot The Differences" as your retinas slowly fade to light gray spots over the next couple decades.

- Here's some Github dorks that have been especially effective in my personal experience.

- `filename:sftp-config.json password`
- `filename:.env MAIL_HOST=smtp.gmail.com`
- `extension:sql mysql dump`
- `filename:wp-config.php`
- `filename:.env DB_USERNAME NOT homestead`
- `filename:credentials aws_access_key_id`
- `filename:.s3cfg`

For tools that automate this process, the most popular tool is probably **Trufflehog** by TruffleSecurity, which has been getting better and better and continues to be updated pretty regularly. It is pretty helpful as it scans through all of the past versions of a repository, which would be a nightmare to do manually. You can grab a release here:

https://github.com/trufflesecurity/trufflehog/releases

Search across all versions of a specific repo :

python3 trufflehog.py --regex --entropy=False https://github.com/<yourTargetRepo>

One thing that hurts me physically when I use the entropy method is an avalanche of false positives. I am really glad they eventually added the feature to turn this off because a lot of useless, but random-looking blobs like checksums, commit hashes referring to git submodules, file hashes used for storing mega-files in git-lfs and git-annex and very boring public keys) get swept up in the results in nauseating amounts. Anyways, you can turn off entropy if its heuristics are also poisoning you and you can use a search mode that uses regex instead, oftentimes for better results.

trufflehog --entropy=NO --regex {git_repo_path_or_url}

Gitrob is also really cool and comes with a great interface to parse through the data it finds. Since there's a lot of false negatives in any tools in this area (or even through manual exploration), this makes crossing off the useless ones much easier and more enjoyable.

Grab a release here:

https://github.com/michenriksen/gitrob/releases

You'll also need to designate a Github API token via this syntax. Place it into the end of your .zshrc file or in your user's .env file.

export GITROB_ACCESS_TOKEN=GITHUBTOKENHEREBOY

Basic usage looks like this:

gitrob -org ORGANIZATIONYOUWANNADESTROY

gitrob -repo https://github.com/weaksauce/badrepo

Jason Haddix also has an awesome script that will spit out a bunch of links to Github searches with the dorks pre-ordained throughout them. This beats typing them and can lead to some easy wins.

https://gist.github.com/jhaddix/1fb7ab2409ab579178d2a79959909b33

Here are some links to other resources or tools to aid in your conquest.

Lists of dorks

https://github.com/techgaun/github-dorks

https://github.com/random-robbie/keywords/blob/master/keywords.txt

Other Tools:

https://github.com/BishopFox/GitGot

https://github.com/Talkaboutcybersecurity/GitMonitor

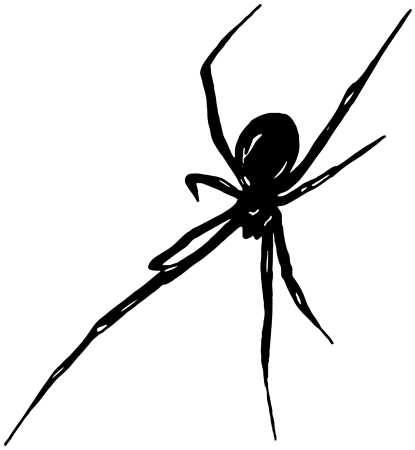
https://github.com/michenriksen/gitrob

https://github.com/tillson/git-hound

https://github.com/kootenpv/gittyleaks

https://github.com/awslabs/git-secrets

https://git-secret.io/



Subdomains

I am almost certain you know what these are so we can just get to the actionable and useful commands that act as the shiny coins on the silver crumbly part of the scratch tickets that are an organization's attack surface. This will definitely be the part of recon that will yield the most attack surface and as such, has the most techniques, tools and competition involved. There are teams of people and proprietary services that will probably always beat us in terms of endless hammering to discover subdomains, so the secret to finding things that others haven't will come down to creative wordlists, obscure services, making sure your API keys are filled out and anything creative or non-automated that you can work out. Also, chaining unique combinations of tools can produce results that others haven't and figuring out how the developers behind the domain think and what shortcuts they may have taken, at the cost of security.

Passive Subdomain Enumeration

Passive enumeration refers to methods that do not disturb or interact with the target server in any way. This can be useful in OSINT when dealing with very paranoid targets or other highly sensitive situations, where your target's behavior may be altered by being alerted of any unexpected attention on their end. Essentially, we will be querying publicly available sources, databases, certificate transparency logs, online code databases, and online services to find these subdomains.

Passive Subdomain Discovery with Shodan

Shodan is better utilized for other purposes but you can still sometimes get some cool subdomains from it. You can do it either through the command line tool, mentioned earlier via:

shodan domain target.com

You can also gather domains with a little more success and much kinder output with the **shosubgo** tool by incogbyte. For whatever reason, it sometimes pulls a few more subdomains than the regular Shodan tool. It's baffling but acceptable. It can be installed via:

go install github.com/incogbyte/shosubgo@latest

Or grab a pre-compiled binary (virus-free on foe nem)

https://github.com/incogbyte/shosubgo/releases/

Attack the domain you hate the most:

shosubgo -d bridge9.com -k stolenshodanAPIkey

One other tool I'd like to mention here is **wtfis** by pirxthepilot. It uses a handful more API keys than just Shodan but it promises to use as few as possible of your monthly queries and tries to use the community/free tiers as much as possible. You can find the repo at https://github.com/pirxthepilot/wtfis or you can install it via python's pip:

```
pip install wtfis
```

Set as many of these environmental variables as you can for maximum useful output from the tool.

- `VT_API_KEY` (required) - Virustotal API key
- `PT_API_KEY` (optional) - Passivetotal API key
- `PT_API_USER` (optional) - Passivetotal API user
- `IP2WHOIS_API_KEY` (optional) - IP2WHOIS API key
- `SHODAN_API_KEY` (optional) - Shodan API key
- `GREYNOISE_API_KEY` (optional) - Greynoise API key
- `WTFIS_DEFAULTS` (optional) - Default arguments

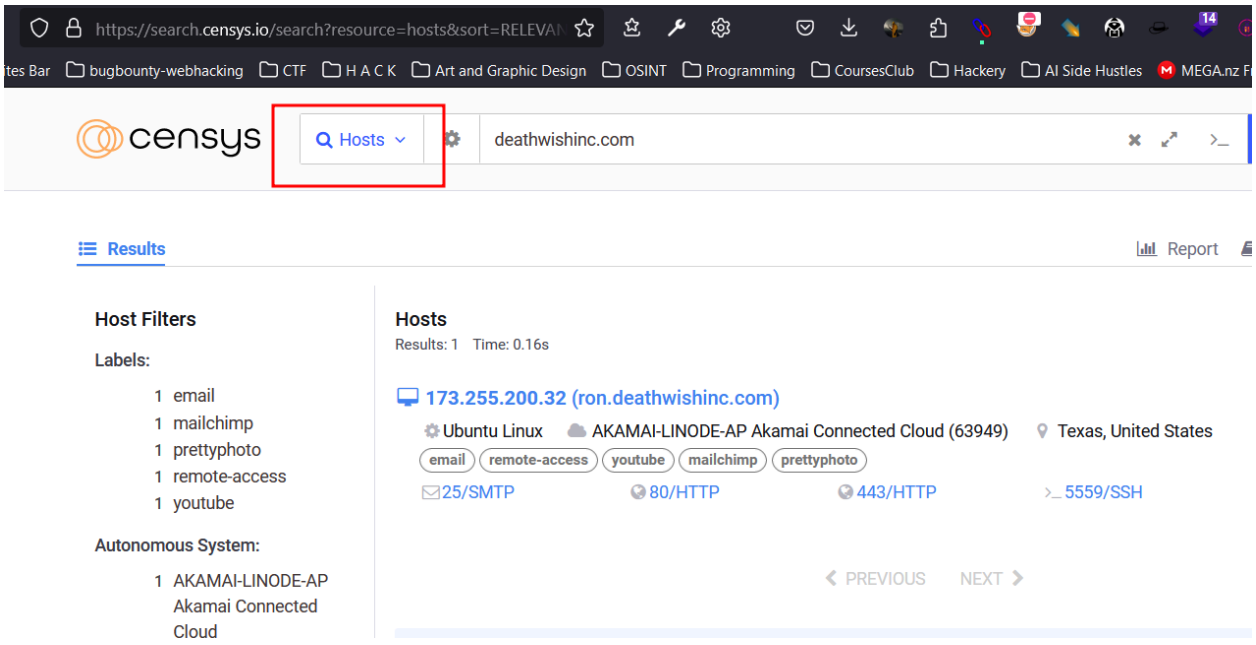
Usage is simpler than set-up, with the following syntax to find what you want.

```
wtfis FQDN_OR_DOMAIN_OR_IP
```

Passive Subdomain Discovery with Censys

Censys is a service similar to Shodan that scans the entire possible range of addresses across the top 3,500 ports and indexes the data. You'll need a free account to use search filters and operators. You can access it via its web interface at https://search.censys.io

The easiest way to get started is to search for your domain in the "Hosts" database



You can also combine search terms with simple boolean operators.

```
parsed.names:skinsuits.com and tags.raw:trusted
```

You can also get a pythonic version of Censys for the command line.

Install it via:

```
python3.11 -m pip install censys
```

To use this in a productive way, you'll need to grab your pair of API keys and enter them into the tool.

```
censys config
```

The simple way that the Censys command line tool provides a list of subdomains these days is with the following simple syntax.

```
censys subdomains oldmilk.org
```

Since nothing can truly ever be free, Censys leaves us with a torturous output that is not compatible with any tools that exist today. So clean her up gently like this.

```
censys subdomains cozy.tv | choose -1 | sort -u | tee
censys-subs-cozy.tv
```

A Tactic Fit Exclusively For A Gentleman

If you have an IP within your target's CIDR and it doesn't resolve, you could also search for that IP directly and check its history through the web interface to potentially discover some forgotten subdomains.

To search by organization name, you can get a fat JSON file with a lot of information, including ASN, IP, open ports and their associated services, certificate hashes and some more stuff that isn't super important for what you're probably doing.

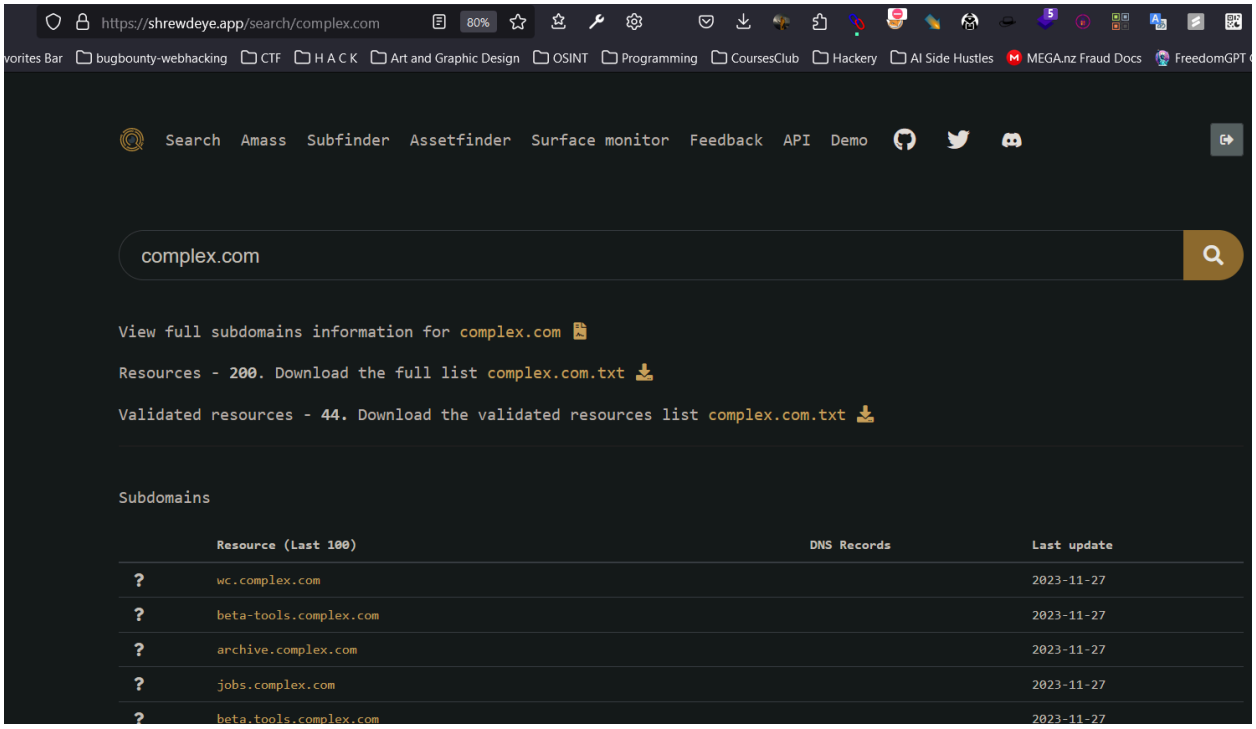
```
censys search 'services.http.response.html_title:
"Facebook"' --index-type hosts
```

Similarly, you can get Censys to filter out non-living subdomains and IPs through the following command.

```
censys search 'services.tls.certificates.leaf_data.names:
cannedhorsemeat.com and
services.http.response.status_code="200"'
```

Takin' it Easy with ShrewdEye

Shrewdeye (https://shrewdeye.app) is a killer service that lets you cut corners (speed, simplicity and opsec) while also getting better results (and parsed in the output you deserve) and it is rumored that they even do it for free. Shrewdeye stores lots of scheduled subdomain scans via tools like Assetfinder, Subfinder and Amass into their backend, which can be queried and parsed into a finely aggregated text file. After you enter your target domain, you can also have Shrewdeye check to see if any of the subdomains are resolvable and available online, right before your eyes, blessing your IP from punching these subdomains and potentially getting banned. If the domain is not in Shrewdeye's database, it also starts firing off the three aforementioned tools so it will have it for you shortly.



Finally, Shrewdeye has one very awesome feature that every hunter would be using if they knew about it. Under “Surface Monitor”, you can drop a list of wildcard domains that you are tracking and receive an email notifying you when anything new shows up. I’m sure many of you had a tool, bash script or cron job that was supposed to do exactly this but somehow it disappeared and you didn’t really care too much to fix it. Well, now you can truly *set it and forget it* for **free.** Enjoy this feature and don’t tell too many people about it, because once they start hitting their monthly email limits, they may not be so recklessly liberal with the availability of these tools.

Third-Party One-Liner Blasters

Here’s a list of some one-liners you can use to query various third-party passive subdomain enumeration services or slap them all into a bash

script. These are all public and available to anonymous access, so you won't need to sign up or use an API key.

Querying the elusive Riddler.io:

```
curl -s "https://riddler.io/search/exportcsv?q=pld:YOURDOMAINHERE.COM" | grep -Po "(([\w.-]*)\.([\w]*)\.([A-z]))\w+" | sort -u | tee riddler.txt
```

Wayback Machine / Archive.org Subdomain Pull:

```
curl -sk "http://web.archive.org/cdx/search/cdx?url=*.$domain&output=txt&fl=original&collapse=urlkey&page=" | awk -F/ '{gsub(/:.*/, "", $3); print $3}' | sort -u | tee waybacksubs.txt
```

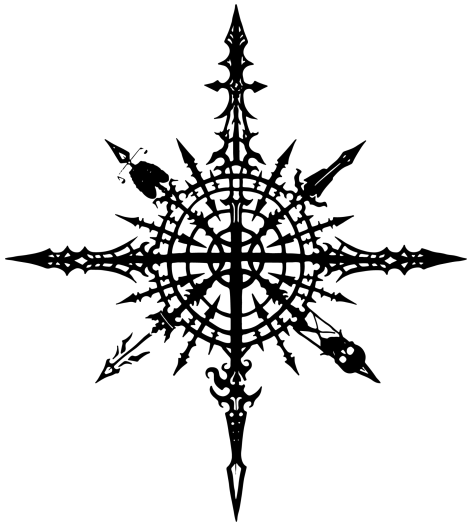
JLDC

```
curl -s "https://jldc.me/anubis/subdomains/$domain" | grep -Po "((http|https):\/\/)?(([\w.-]*)\.([\w]*)\.([A-z]))\w+" | sort -u | tee jldc.txt
```

```
curl -s <https://rapiddns.io/subdomain/example.com?full=1> | grep -oP '_blank">\\K[^<]*' | grep -v http | sort -u
```

The penultimate source for search engines that may pique your interest and lead to the nuclear devastation of your target organization was put together by edoardott in the form of **awesome-hacker-search-engines**. You could do all your recon in the browser and if you find some way to rip through a good chunk of these with automation, scraping and scripting, then god bless your heart (and please share it with me). There are untold hidden treasures lurking here and the best of you will discover them. Heed this warning with sincerity : be careful with this link, it is quite easy to open 666 tabs and spend all afternoon here.

https://github.com/edoardottt/awesome-hacker-search-engines



Certificates

To increase security by providing a way to validate the legitimacy of a domain and help curb phishing and impersonation, SSL certificates are acquired by domains from registered certificate authorities (CAs). In order to be able to effectively monitor and audit these certificates, they are recorded in a cryptographically secure manner, then published and accessible to everyone via Certificate Transparency Logs. These logs exist to be monitored proactively and allow domain owners to discover if fraudulent certificates are issued for their domains, while also keeping certificate issuing entities accountable, as anyone can trace these timestamped certificates granted to malicious entities back to the carefree CA and browbeat them into obscurity.

Conveniently for us, we can utilize this to monitor certificates ourselves or see their entire backlog of them to enhance our recon efforts. On the other hand, since logs are never removed, there may exist a lot of crap

subdomains to parse out that haven't existed for a decade, but that's nothing we aren't dealing with every minute in this game.

It would be remiss not to mention *crt.sh* first. You can view the site in your browser and use the search function (which accepts wildcards using a "**%**", like %.kindboys.com) or query it directly using the syntax below (with your own domain as a value, obviously).

https://crt.sh/?q=onlyfans.com

Because of its unlimited, unauthenticated free tier, constantly updated and a profusion of historical records dating back to the invention of the internet, **crt.sh** is generally regarded as a grounded, excellent source for certificate recon for hunters like me and you. To get a parsed list of subdomains from the terminal, you can use the command below, then pipe the output to a file for your records.

```
curl -s https://crt.sh/\?q\=%25.onlyfans.com\&output\=json |
    jq . | grep 'name_value' | awk '{print $2}' | sed -e
 's/"//g'| sed -e 's/,//g' |  awk '{gsub(/\\n/,"\n")}1' |
                        sort -u
```

Certificates with Censys

You can also use **Censys**, which claims to hold the largest collection of certificates at over 5 billion certificates logged. Pro accounts are marketed to organization's for monitoring their own resources, so the free account gets you total access to their data.

. From the web UI at "https://search.censys.io/", you can set the dataset to "*Certificates*" instead of "*Hosts*" and search for your target domain.

Through the command line, here's a few queries to execute to increase your target's attack surface (just replace my stupid domains with your real ones).

443.https.tls.certificate.parsed.subject.common_name:poundtown.com

[443.https.tls.certificate.parsed.extensions.subject_alt_name.dns_names:[fromdariver2daC.com]]

A tool that does a lot of special stuff under the hood and presents you with a nice list of IP addresses coupled with their open ports, ready for further investigation, is projectdiscovery's **uncover**. You can conjure the binary yourself via the method below, which will also set you up with a default config file in the expected location. Otherwise, you will have to create it yourself if you go to their releases page, and there's a lukewarm chance something will go very wrong (the location it should be found at is `$HOME/.config/uncover/provider-config.yaml` though).

go install -v github.com/projectdiscovery/uncover/cmd/uncover@latest

Uncover works by giving it a query and it will work its mysterious magic and give you some endpoints; their usefulness ranging across a broad

spectrum of value and their origin highly confounding. You'll want to fill out as many API keys as you can for this tool, as most of these providers will not return anything without them. By default, it checks Shodan, but also has the ability to find hidden attack surface assets through Fofa, Censys, Zoomeye, Hunter, Quake, Netlas, CriminalIP, PublicWWW and HunterHow.

Specify your provider or providers with the **-e** flag and your query with the **-q** flag. Queries can be any variety of things from domains, technologies (i.e. "Jira", "Nginx"), organizations or even a file with a list of dorks or subdomains.

```
uncover -q 'tay-k47.com' -e
shodan,censys,fofa,quake,hunter,zoomeye,netlas,criminalip

uncover -q 'ssl:"Uber Technologies, Inc."'
```

You can even use it for passive port finding via the third party services it queries (not including an engine defaults to Shodan).

```
echo 51.83.59.99/24 | uncover
```

Finally, a hyperfast and direct way to chop up, flip and serve subdomains from a TLS certificate is glebarez's **cero** tool. Embodies the Linux ideals of doing one thing really well and produces results ready for any pipe you decide to send them through.

```
go install github.com/glebarez/cero@latest
```

This tool takes domains, IPs, IPv6, CIDR ranges and even non-HTTP protocol targets as inputs and fires back like an overactive chain gun shooting the shields off an organization's attack surface.

```
cero whitecastle.com
```

```
cat deez_subs.txt | cero -c 500
```

```
cat cidr-ranges.txt | cero -p 4443,8443,2083 -c 1000 | anew cero-cidr.txt
```

I'd like to take a quick detour to talk about tomnomnom's tool **anew.** It is super simple, super clever and super useful, as I use it every ten minutes during every session.

As much as I love using "**tee -a "** (show output and append output to file), 90% of the time, I need to follow it with a couple more commands to remove duplicates and route the output to another file, then overwrite my old file with the new file**.** This tool condensed that workflow into a single command, only appending new data to your output. Just like **tee**, it will create a file if one doesn't exist already.

Use cases for this are overwhelmingly frequent - essentially, any time I need to update a list of subdomains, URLs, JS files, IP addresses.

```
vita -d goatpig.org > goatpig-subs.txt
```

```
python3.11 subj.py goatpig.org | anew  goatpig-subs.txt
```

The output will show only the newly discovered subs. You could also utilize this by piping it to a file and setting it as a cron job, so you could see which subdomains were newly discovered, or routing it directly to notify.

```
subscout -t goatpig.org -o subscoutput.txt
```

```
cat subscoutput.txt | anew goatpig-subs.txt | notify
```

If you want to save a few more milliseconds of each day, you can unsmooth your brain a little bit more with dwisiswant0's tool **unew** (found at https://github.com/dwisiswant0/unew). It combines **anew** and **"sort -u"**, in addition to being able to condense multiple of the same URLs with different parameters into a single URL with all of the parameters. This is super helpful when you have a christload of URLs that are all extremely similar padding your passively gathered URL files.
So if this is your URL file:
http://whitepigs.org/comatose?id=200
http://whitepigs.org/comatose?admin=noteven
It can compress it into:
http://whitepigs.org/comatose?id=200&admin=noteven
The command will look like:
`cat whitepigurls.txt | unew -combine > whitepig-adv.txt`

The **Content Security Policy (CSP)** header serves as a security protection against XSS or other malicious content by defining and restricting the locations that code is allowed to load from. Rather than

hosting all their content on one server, most modern websites pull their resources across cloud providers, image hosting services, content delivery networks and official Javascript libraries, in an effort to deliver content faster through the parallelization capabilities of browsers. Setting the CSP header to trusted locations is commonly misconfigured, in addition to some "trusted" services offering anyone the ability to host arbitrary code, can make the CSP a minor inconvenience for many malicious actors. However, this is a book about recon and the assets often listed in the CSP headers can illuminate more assets that serve to increase our opportunities for finding vulnerabilities against our target.

Let's get straight to the point with a tool that gets us there quickly - 0xbharath's tool **domains-from-csp**. Without flexing your head too hard, I'm sure you understand that it does a thing and spits out domains from the CSP. Install it via the following instructions:

```
git clone https://github.com/0xbharath/domains-from-csp

cd domains-from-csp

python3.11 -m pip install requests click
```

The tool takes a full URL as its first argument for input and lovingly add "**-r**" to resolve the output as well.

```
python3.11 domains-from-csp.py https://slimetimelive.com -r
```

You can also directly pull the contents of the CSP header using curl, for double-checking or integrating into your automation scripts.

curl -s -I -L “https://www.newyorker.com/" | grep -Ei ‘^Content-Security-Policy:’ | sed “s/;/;\\n/g”

Our good boy here also has another tool that compliments and reflects the prior script. 0xbharath’s **assets-from-spf** earns its namesake and delivers exactly what we deserve, some assets from our target’s SPF (Sender’s Policy Framework). Prepare your terminals, my friends.

```
git clonehttps://github.com/0xbharath/assets-from-spf

cd assets-from-spf

python3.11 -m pip install click ipwhois
```

Fire it off with this simple syntax (add an **-a** flag if you want ASN included in the output as well).

```
python3.11 assets-from-spf.py morbidsniper.gov
```

One other field that can help you find subdomains, as well as associated domains for your target (lovingly referred to as horizontal enumeration by some), is the “Subject Alternative Name” (SAN) field. This field allows domain owners to add multiple hosts on a single SSL certificate, so they are generally filled with locations where the domain’s content is hosted and other domains owned by the same organization.

Here’s a tool by projectdiscovery called **tlsx**, because we might as well mention everything they’ve ever made in this book. You can grab it via

their “Releases” page (https://github.com/projectdiscovery/tlsx/releases) or punch in the following:

```
go install github.com/projectdiscovery/tlsx/cmd/tlsx@latest
```

There’s more features than nightmares that geniuses like myself suffer through in a week, but to fulfill our most common recon needs you can pull relevant SAN data and common name (CN) data via the **-san** and **-cn** flags.

```
echo 173.0.84.0/24 | tlsx -san -cn -silent
```

To restrict the output to a list of domains, you can add the **-resp-only** only. You can make a cool chain with **dnsx** to resolve your output and filter out invalid locations, followed by **httpx**, to add some more information, like the site’s title, server type, basic technology and status code.

```
echo 173.0.84.0/24 | tlsx -san -cn -silent | dnsx -any -resp-only | httpx -td -title -cl -sc -ip -server -cname -asn -cdn
```

To wrap up certificates real quick, here’s a couple browser based sources you can check out if you’re tired of the terminal.

https://ui.ctsearch.entrust.com/ui/ctsearchui

This service is neat because you can visually see if the certificate is under a cloud CDN immediately and you can focus on culling the weakest gazelles in the herd. This service is also known for being a big snitch, listing off which WAF is being used and other details without even being pressed too hard! This isn't even dry snitching, there's no plea deal on the table, *entrust.com* isn't even charged with anything!! Absolutely mind-boggling snitching we are witnessing here, just absolutely disgusting stuff.

And here's just a few more resources if you aren't satisfied or feeling physically ill. Maybe some services are slacking on keeping these updated and organized, so be proactive and find a favorite you can trust.

https://developers.facebook.com/tools/ct/

https://google.com/transparencyreport/https/ct/

Passive Subdomain Enumeration Tools

The absolute and undisputed Chad of passive subdomain enumeration at the moment is projectdiscovery's **subfinder**, as evidenced by its notoriously high prevalence in all recon frameworks and its absolute blast to the (Github) stars as soon as it was introduced. The tool is

commonly underutilized by smooth brains that fail to add as many API keys as they can find, afford or steal from the internet. There's also a couple arguments you shouldn't miss that can be found below.

Releases can be found here, but I think it is better to install it as it will automatically create the correct default location for your config file.

https://github.com/projectdiscovery/subfinder/releases/

Install via Go, the preferred method for the privileged ones like ourselves.

```
go install -v
github.com/projectdiscovery/subfinder/v2/cmd/subfinder@lates
t
```

Usage should look like the command below. Don't forget to use a freshly updated list of resolvers and use the **-all** and **-recursive** flags for maximum output.

```
subfinder -all -recursive -rL ~/resolvers.txt -d
commanderbyrd.com
```

Another very advanced and impressive tool is OWASP's **amass,** which has a handful of varied and unique features, including the ability to conjure a visual representation of your data, storing a database of past searches (which is useful for comparing searches over time and discovering which subdomains are newly active) and even possessing its own scripting language, giving users the ability to make it extendable with new features. Like the last tool, it is crucial that you

deflower the config file with as many API keys as possible, as it will deliver tenfold the results compared to a virgin copy of the tool. The default location that amass will search for your config file is "***$HOME/.config/amass/config/config.ini",*** in case something is wonky on your end and you need to fix it.

The "intel" feature does a more OSINT-related type of passive enumeration, with the potential of extending beyond just subdomain enumeration and potentially discovering additional CIDR ranges, ASNs and related domains. In addition to the **-org** flag, you can set it to an ASN with **-asn** (i.e. **-asn 6107**), a domain with **-d** (i.e. **-d doomburgers.com**), a CIDR with **-cidr** (i.e. **-cidr 10.9.1.0/21**) or any combination of those.

amass intel -org "Bacon Boys, Inc."

Once settled on a single domain to enumerate, you can move on to the "enum" feature and set it with the **-passive** flag for passive subdomain enumeration purposes.

amass enum -passive -d baconboys.org

amass enum -passive -config -d baconbros.gov -o amassivelyniceoutput.txt

Amass being exceptional in your results is heavily dependent on the APIs you insert into your config. To see a list of all the API keys that are being utilized and which ones you are missing, you can run:

amass enum -list

Definitely make it your goal to find ‘em all, the free ones, at least. It will literally triple your output here.

The third megalith of passive subdomain aggregation tools I’ll mention here is **findomain** by... findomain. I won’t retread the same material here, let’s just get to it. Download a binary of your choosing here and *don’t forget* to fill out the API configuration file, I am really serious about this.

https://github.com/Findomain/Findomain/releases/

For the big boss power command, see below.

```
findomain -t stonecold316.net --external-subdomains --ip
--http-status  --lightweight-threads 100  --http-retries 2
--resolvers ~/resolvers.txt
```

If you want to be a unstoppable madman, you can also add **--pscan** for additional port scanning, **-s** for screenshots

Shifting gears a little bit, this next tool parses all Github repositories to uncover mentions of subdomains related to your target. This brilliant piece of code is known as **github-subdomains** by gwen001, who has enriched my toolkit with many awesome tools based on clever ideas

that fulfilled needs I didn't know existed. Install the tool via the following command:

```
go install github.com/gwen001/github-subdomains@latest
```

This tool requires you to have an active Github API token, either in a file (listing one per line, if you have more than one to use) or via set in your Linux environment. You can set it for your current session via the following syntax.

```
export GITHUB_TOKEN=token1,token2...
```

The syntax to use it is pleasantly simple too.

```
github-subdomains -t ~/githubtokens.txt -d longpigs.gov
```

Gwen001 also has another similar tool, although it returns much less output. Albeit, it is mentioned here because in the chance that it finds something, it will likely be the only tool with that finding and we are going for total uncompromising comprehensiveness of our recon methodology. It does the same thing as the Github one above, but targets Gitlab instead.

```
go install github.com/gwen001/gitlab-subdomains@latest
```

Add your Gitlab API tokens (or steal them) and run it with the same syntax as above.

```
gitlab-subdomains -t ~/gitlabtokens.txt -d doormouse.org
```

Listed ahead are some lesser known tools that consistently add something new to my automated script of around twenty different

subdomain enumeration tools. The past few are obviously great but since everyone knows about them, they won't uncover the most obscure paths that you are likely seeking. I have spent countless hours grinding through Github tools with a seldom few stars to present you with this information. Let's fucking go.

Hueristiq is one of my favorite new creators and every tool he has put out has the gold standard of excellence and adds something genuinely unique to the standard bug hunting toolkit. His passive subdomain enumeration tool, xsubfind3r, is no exception and it has added subdomains after running multiple frameworks against a target consistently.

Grab a compiled binary version here:

https://github.com/hueristiq/xsubfind3r/releases/

Or do it yourself.

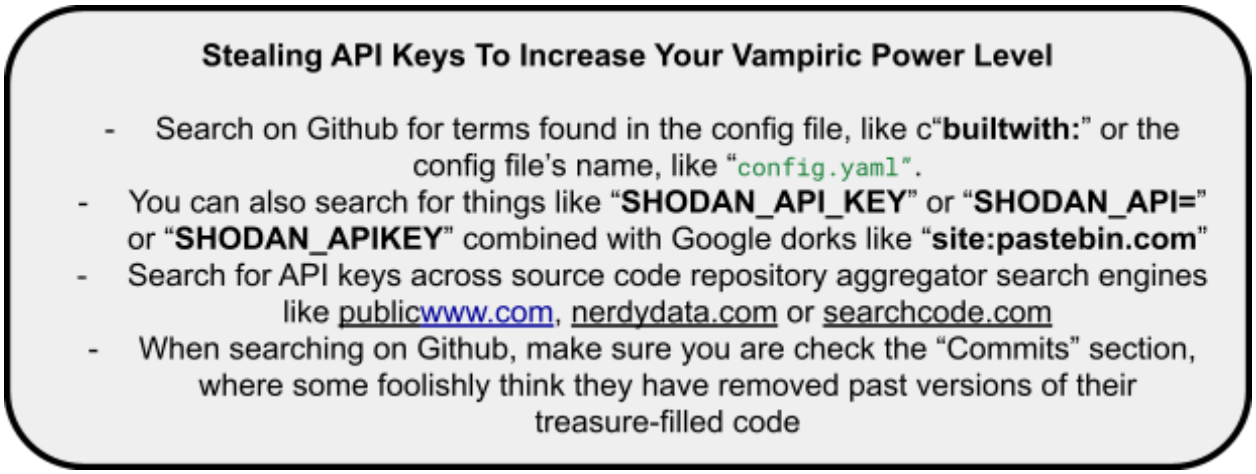
```
go install -v
github.com/hueristiq/xsubfind3r/cmd/xsubfind3r@latest
```

Installing the tool through this method has the added benefit of automatically creating a config file for API keys, which is an absolute necessity if you want to have self-respect while doing these things. If you grab the binary on its own, you will want to head to Github and grab the config file, place it in the default location of "**$HOME/.config/xsubfind3r/config.yaml**". The majority of them have free tiers, although there are a couple that cost money, like intelx.io and some that are special and elusive, like chaos.projectdiscovery.io (invite

only, from the clever demons on the project discovery team). Signing up for these doesn't take long and it will probably double the amount of results you get from using the tool. For API keys that aren't easily attained, you can try to steal them from people that accidentally posted them online.

You can use the tool like so:

xsubfind3r -d militant.zone -t 66 -o xsubfind3r-output.txt



Another great one that I've recently discovered is **puncia** by ARPSyndicate, which claims to be "The Panthera Puncia Of Cybersecurity". It holds a bunch of exploits within that it will readily unload on an unsuspecting unpatched victim but it is quite adept at finding subdomains…. Passively that is.

```
python3.11 -m pip install puncia
```

The syntax always begins with **"puncia subdomain**", followed by your target domain and then the output file you want to save it to.

puncia subdomain pantherskin.com pantherskin-subs.txt

Bufferover.run is a commonly queried API Here's another one-liner you can use if your smuggler's run is rewarding at the expense of some sad human who is unable to stop leaking bufferover API keys.

Free tier:

curl 'https://tls.bufferover.run/dns?q=.YOURDOMAINHERE.com' -H 'x-api-key: <your api key here>'

Paid Tier

curl "https://bufferover-run-tls.p.rapidapi.com/ipv4/dns?q=.YOURDOMAINHERE.com" -H "x-rapidapi-host: bufferover-run-tls.p.rapidapi.com" -H "x-rapidapi-key: YOUR_FREE_KEY_HERE"

Dnsdumpster is a neat source that provides subdomains that aren't found elsewhere sometimes. Pulling the information isn't too easy, as it doesn't have a super clean API to hit but you can use the script below to do so and pipe the output into your records or any other tool. Save the following to a script, grant it executorial powers ("**chmod +x dnsdump.sh**") and then run it via "**dnsdump.sh domain.com".**

host=$1

```
# extracting the CSRF token so our request doesn't get killed

r=$(curl -i -s https://dnsdumpster.com | grep csrf)

csrftoken=$(echo $r | grep "csrftoken" | cut -d "=" -f 2 | cut -d ";" -f 1)

csrfmiddlewaretoken=$(echo $r | cut -d "=" -f 12 | tr -d '">')

subdomains=$(curl -s -X POST -H "Referrer: https://dnsdumpster.com/" -H "Cookie: csrftoken=$csrftoken" -d "csrfmiddlewaretoken=$csrfmiddlewaretoken&targetip=$host&user=free" https://dnsdumpster.com | grep "<tr><td class=\"col-md-4\">" | grep $host | grep -Po "(([\w.-]*)\.([\w]*)\.([A-z]))\w+")

echo $subdomains | sed 's/ /\n/g'
```

Active Subdomain Enumeration Tactics

When we no longer feel the weight of the ironwrought boots of paranoia and we abandon our interest in being a cybernetic ghost, seldom leaving a drop of ectoplasm across their network packets, we can decide to bully the target ourselves a little bit. We use active

techniques to do things that no one has ever done before to create new dynamic options for potential subdomains (or at least, not published it openly and indexed it into a search engine), we want freshly enumerated results or we want to test (and uncover) a naming pattern we have noticed our results from the passive subdomain testing. .The main reason, of course, is to increase our attack surface and find things that have been intentionally obscured, ignorantly forgotten or otherwise. These barren wastelands that once served packets en masse to the world tend to be more likely to serve strange and ugly custom-code projects from decades ago, sites that grab your attention for their painfully low-resolution graphics or almost hateful use of colors, in a way that deeply wounds people who enjoy seeing contrast when they read and subjects the color blind to virulently inequitable access to information. As with any path you take, but significantly more evident in cybersecurity fields,

First, let's talk about zone transfers. Simply put, this occurs when the data (domain names, IP addresses and more) from one DNS server's zone is copied onto another, secondary DNS server. Utilizing the zone walking technique can uncover hidden or forgotten subdomains, internal targets without official DNS routing and shed heavenly light onto the infrastructure of your target.

The following tools come with every Linux system (and Windows) that I've tried so you should have them. If not, I will pray for you.

Replace "zonetransfer.me" with your target domain.

With **dig**:

```
dig ns zonetransfer.me +noall +answer

dig axfr @nsztm1.digi.ninja zonetransfer.me
```

With **host**:

```
host -t ns zonetransfer.me

host -t axfr zonetransfer.me intns1.zonetransfer.me
```

With **nslookup:**

```
nslookup -type=ns zonetransfer.me

nslookup -query=AXFR zonetransfer.me name.server.here
```

Since we already have **subfinder**, you can use it for brute-forcing domains by using the **-b** flag to enable it and the **-w** <WORDLIST.txt> flag to specify a wordlist to use.

```
subfinder -d freelancer.com -b -w jhaddix_all.txt -t
100 --sources censys --set-settings CensysPages=10 -v
```

Amass has also been mentioned previously in this book, but here is how to really push it to the pavement.

amass enum -d streetsharks.com -active -o amass-active-streetsharks.txt -brute -alts

These flags will also show you the source of where the subdomain was pulled from, the IP that each subdomain is connected to, perform brute forcing of subdomains and also perform alterations on them. Amass is pretty great as it keeps a database of findings and hit tons of APIs but it isn't the fastest bruteforcer.

For bruteforcing subdomains, the best tool I've found yet is **puredns** by d3mondev. Besides being really fast, it does a couple other cool things like removing all the unusable characters from your wordlist and switching everything to lowercase, detects wildcards, built on top of massdns to query a stupid amount of resolvers and then validates the final results instead of spewing crap all over your screen. The validation process is actually a DOUBLE validation from trusted resolvers, so it avoids all those poisoned ones out there looking to do bad stuff to good people.

Download a binary here.

https://github.com/d3mondev/puredns/releases/

Or build it yourself via:

go install github.com/d3mondev/puredns/v2@latest

Run with like so:

puredns bruteforce subdomain_list.txt domain.com -r ~/resolvers.txt

You can also get a really good list of resolvers via this command:

wget https://raw.githubusercontent.com/trickest/resolvers/main/resolvers.txt

If you need good and massive wordlists to feed into this tool, look no further than the links below.

Assetnote is one of the best security researchers in the bug bounty space and does some seriously good work. They have created a huge file of potential subdomains as a result of their tireless scanning across the internet. It's called "best-dns-wordlist" and contains about 9 million entries.

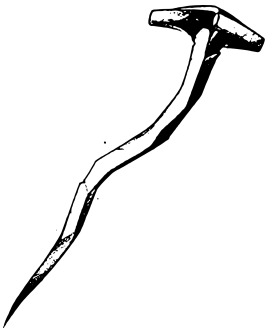
https://wordlists-cdn.assetnote.io/data/manual/best-dns-wordlist.txt

Next, n0kovo created this wordlist by scanning the entirety of the IPv4 space across the world and pulling all the subdomain names from every TLS certificate in existence. It performs quite well and is around 3 million entries long.

https://github.com/n0kovo/n0kovo_subdomains/blob/main/n0kovo_subdomains_huge.txt

If you need a smaller wordlist for shorter sessions or not using a VPS like you should. You can try this good one by six2dez, at about 100,000 entries long.

https://gist.github.com/six2dez/a307a04a222fab5a57466c51e1569acf/raw



VHOSTS

Virtual Hosting (VHOSTs) allows multiple domains to share their existence across a single server. Organizations opt into this method as a way to reduce resource overhead, save on costs and ease of management. Having one server support these domains makes applying changes and fixing problems a bit easier, as all the content is accessible in one place, while also being isolated enough that their effects on each other are insignificant.

VHOSTs can be configured onto a server in a couple different ways. First, each domain may have its own IP address and when the server hosting them receives a request, it can properly route the browser towards the correct resource. The client cannot really interfere with this process, so it is safer, but it adds to overhead on the organization's side, as they will need to continuously obtain IP

addresses for each domain they own and the associated work involved with the continental internet registries that reign over them.

Therefore, many organizations use a name-based virtual host configuration setup instead. In this configuration, the server will look towards the "`Host`" header received in the request to accurately route the user towards their desired content. Through various misconfigurations and differences in server software, all kinds of interesting vulnerabilities can occur when the "`Host`" header is manipulated, leading to the server sending the user and their tainted data into their internal network (SSRF) or mixing up requests due to incongruencies between the front end and back end, leading the server to give you another user's data (HTTP Request Smuggling).

The most prominent way of finding new VHOSTs and increasing your attack surface involves using a wordlist to fuzz subdomains and comparing the response from the server. While this may sound like DNS brute-forcing, there is no DNS server involved since subdomain resolution is based on the server configuration and utilizes the HTTP or HTTPS protocols, rather than DNS. When HTTPS is involved, the SNI (Server Name Integration) is an extension of the TLS, which tells the server which domain name it is attempting to access.

Due to the various configurations, VHOSTs can be a little wonky to detect and to do a really comprehensive check, you can run a manual check, as well as a few automated tools. So many times, I've run two tools with the same purpose and found different results, as they use different methods, query different APIs or data sources or sometimes

one just sucks and the developer should be punished and stripped of any respect. So, let's say you find an IP within your target's owned IP range and want to test it yourself to see what it's hosting (generally, port 80 is a redirect to 443 so I jump right to it, but you know, be as thorough as you feel comfortable with). At the time of this writing, Caterpillar Inc. is a public program on HackerOne with a wicked huge scope, so feel free to punch their assets as hard as you want until that changes (FAIR USAGE and SAFE HAVEN).

So we check the IP address and verify that it is owned by our intended target.

```
nslookup 165.26.57.0

0.57.26.165.in-addr.arpa   name = aemha-intweb.cat.com.
```

The output from nslookup gives us the green light to probe further. However, we get served with a 404 when we try to check the site.

```
curl -i -k https://aemha-intweb.cat.com

HTTP/1.1 404 Not Found

date: Sat, 20 Jan 2024 06:26:41 GMT

server: Apache/2.4.6 (Red Hat Enterprise Linux)
```

We can try messing around with things here in a few ways. We can try to pull the URL by using its real IP address and see if we get a different response. This will help us to verify that the server is storing multiple sets of content and using information from the request to decide where to send the client.

```
curl -i -k https://165.26.57.0

HTTP/1.0 503 Service Unavailable

pragma: no-cache

cache-control: private, max-age=0, no-cache, no-store

content-type: text/html
```

This is one spot that many pentesters arrive at and believe they have exhausted their options on this endpoint. You can continue probing this with a nice list of potential subdomains that you can insert into the host header and check the responses for differences compared to requesting the IP on its own, as well as deviations from responses sent to the subdomain you are trying to resolve.

So you can automate this a manual way with some bash scripting or use a handful of tools, though I have gotten varied results, so I make sure to use a few tools, double check any positives and manipulate a few things in the requests to see if I can get something interesting to return. Still using the same IP address in the examples above, I added my specified host header into the request.

curl -i -k https://165.26.57.0 -H "Host: signs.cat.com"

Output:

HTTP/1.1 200 OK

date: Sun, 21 Jan 2024 02:58:41 GMT

server: Apache/2.4.37 (Red Hat Enterprise Linux)

And in contrast to just sending a request to the subdomain.

curl -i -k signs.cat.com

Output:

HTTP/1.1 301 Moved Permanently

Server: rdwr

Date: Sun, 21 Jan 2024 02:59:02 GMT

Content-Type: text/html

And checking this subdomain on nslookup points us to another story as well.

nslookup signs.cat.com

Output:

signs.cat.com canonical name = redirect4.cat.com.

redirect4.cat.com canonical name = 86928b2876734aa5bb6e82d75a41b699.v1.radwarecloud.net.Name: 86928b2876734aa5bb6e82d75a41b699.v1.radwarecloud.net

Address: 66.22.73.126

So there is a lot of wonkiness inherent in how people implement these things and the creative configurations they decide on can expose all kinds of things that were meant to be internal. In the case of internal servers, organizations typically have their local network configured to handle the routing and as this content is not intended to be public,

Sometimes your targeted subdomain might look completely lifeless and forgotten.

curl -k -v terminal.ecorp.cat.com

Output:

*** Could not resolve host: terminal.ecorp.cat.com**

*** Closing connection**

Is it really unresolvable?

nslookup terminal.ecorp.cat.com

Output:

Server: 108.61.10.10

Address: 108.61.10.10#53

**** server can't find terminal.ecorp.cat.com: NXDOMAIN**

Trying to probe a little bit more, and interestingly enough...

curl -i -k https://165.26.57.0 -H "Host: terminal.ecorp.cat.com"

Output:

HTTP/1.1 200 OK

date: Sun, 21 Jan 2024 03:17:44 GMT

server: Apache/2.4.6 (Red Hat Enterprise Linux)

x-frame-options: SAMEORIGIN

accept-ranges: bytes

<title>Caterpillar Rocket BlueZone Sessions</title>

It took a while before I even came across some of this VHOST information and how much more enumeration you can actually do. For whatever reason, this is a very underutilized part of the recon process, possibly due to a lack of articles doing a real deep dive and the subsequent lack of awareness of these techniques. Compared to other subdomain enumeration methods, enumeration of VHOSTs is vastly underrepresented in the offsec tooling landscape. I have found that digging into VHOSTs using this semi-manual process to be the most comprehensive method, seldom shared in our circles. You can definitely supplement it with some tools that can start to point in the right directions to try the manual probing just revealed to you. If you find something you want to dig into deeper, you can use *Burp Suite*'s "Match & Replace" rules to force the host header you want into the request or add the desired entry into your **`/etc/hosts`** file.

My favorite tool for this is wdahlenburg's **VHostFinder**. As it was his blog ("https://wya.pl") that first introduced me to this conscious probing of VHOST servers, the multiple steps of enumeration that most penetration testers fall short of performing and a shared belief that the current tools available aren't doing exactly what we needed, in exactly the way we wanted, VHostFinder addresses all these shortcomings and has found me some excellent attack surface that would be remiss to me otherwise. You can install yourself by entering the following into your terminal:

```
go install -v github.com/wdahlenburg/VhostFinder@latest
```

Once you've gotten together a good list of subdomains to test and an IP that belongs to your target, basic usage looks like the following.

```
VhostFinder -ip 10.4.22.15 -wordlist domains.txt -verify
```

If you have a list of IP addresses to test, I circumvent this problem with a little bit of simple bash scripting.

```
for x in $(cat ips.txt ); do echo $x && echo "\n--------\n"
&& VhostFinder -ip $x -wordlist domains.txt -threads 80
-verify  | tee -a vhosthunt-catmassive ; done
```

Absolutely include the **-verify** flag in every scan. Finally, if you have multiple domains belonging to a target and you want to fuzz a list of subdomains across them, you can do the following.

```
VhostFinder -ip 10.10.10.10 -wordlist subdomains.txt -domain
cat.com,solarturbines.com,catwatches.com -verify -threads 80
```

Sometimes, when the target is massive and I already have thousands of valid subdomains connected to my target, I want to fuzz only the valid subdomains to save time. Basically, have a subdomain list without any definite losers and extract the apex domain name from it.

If it is a single domain, you can use **sed**.

```
cat valid-domains.txt |   sed 's/cat\.com$//g'  | tee
extracted-subs.txt
```

Or if it's a mix of domains from the target, use Tomnomnom's **unfurl**, a tool of incalculable value. Grab a compiled release at:

https://github.com/tomnomnom/unfurl/releases/

Alternatively, compile it yourself.

```
go install github.com/tomnomnom/unfurl@latest
```

And use it, as seen below.

```
cat valid-domains.txt | unfurl format %S | tee
extracted-subs.txt
```

Unfurl gets daily use from me, along with Tom's other tools like *anew, urinteresting, httprobe, gf* and more; many of which will make a quick appearance in this tome.

Although VHostFinder does pretty much everything I need it to, it can always be useful to run a few tools, as there are often differences in outcome and I'll mention a couple more here for that reason.

Jobert Abma, co-founder of HackerOne has a neat tool for this that has existed for a while and can be found here:
https://github.com/jobertabma/virtual-host-discovery

It has similar parameters to VHostFinder and requires Ruby to run.

```
ruby scan.rb --ip=192.168.1.101 --host=domain.tld
```

Another tool I've discovered recently that is also really useful, is dariusztytko's **vhosts-sieve** (https://github.com/dariusztytko/vhosts-sieve). This tool automates much of the process for you, which is great, though in doing so, it misses certain vhosts and there is no way to configure some of my gripes into the tool, hence, the importance of running a few tools on your target. The tool works by resolving your input domains list to IP addresses, though unfortunately there is no way to input your own IP addresses list, in the scenario that you are given CIDR ranges in your scope. Next, it divides these domains into resolvable and non-resolvable groups. The tool discerns that the non-resolvable IPs are the candidates it will test for virtual hosts, but misses the chance to find virtual hosts hidden on IPs that do resolve. Overall, the tool is great at checking multiple ports, has an impressive validation methodology and is very simple to use, as it only requires a list of domains to get started. Just keep the manual checks in your methodology and don't get lazy or you'll likely miss something valuable (not every time, but definitely some of the time).

git clone https://github.com/dariusztytko/vhosts-sieve.git

python3.11 -m pip3 install -r vhosts-sieve/requirements.txt

Run the tool with the following syntax:

python3 vhosts-sieve.py -d domains.txt -o vhosts.txt

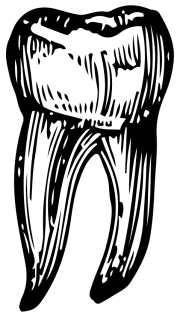
Finally, this isn't directly related to virtual host enumeration but can definitely help point you in the right direction, in case you are dealing with a massive scope, padded with endless subdomains. Hakluke's **hakfindinternaldomains** is an excellent tool with an easily understandable purpose, as just knowing the name of it will already explain what it does. As I mentioned earlier, virtual hosting is sometimes used for an organization's internal purposes, intentionally unresolvable to be obscured from the internet en masse. Employees of the organization will have their **/etc/hosts** file attuned to these hidden gems, as they are intended for usage within the organization only. Often intended for staging or pre-production environments, these assets will usually lack the protections that intentionally exposed ironclad domains are adorned with. Though this tool will point you in the right direction, you will still need to do some work to access them - either through the virtual host exposing methodology explained in this book, finding an SSRF vulnerability, finding a misconfiguration or some other clever way. However, you will be able to fast track that process a bit by knowing which targets to punch at first.

Compile it yourself by entering this into your terminal.

```
go install github.com/hakluke/hakfindinternaldomains
```

Usage is very straightforward, like hakluke's other tools.

```
cat domains.txt | hakfindinternaldomains | tee internal.txt
```



Permutation

Once you have a decent list of subdomains for your target, another technique we can use to uncover esoteric attack vectors is the mutation, contortion and disfigurement of current subdomains. Since there's a gang of tools and they possess some differences in how they vomit forth wild and twisted perversions of your current list, I think it is worth checking out a bunch of them and writing an automation script that will chew and mangle your current list via each tool. Though their internals are varied, their purpose and output is fairly similar, so I won't spend too long talking about them and just hit you with the parts you need to start ripping and resolving your guys.

If you're still a little lost because of my colorful explanations, I'll break it down. You have a subdomain list in a text file, with contents like:

api.wrinkles.org

skinbags.wrinkles.org

shame.wrinkles.org

Then we input our own wordlist, the contents of look like:

dev

morbid

420

The tool will combine these in tons of different ways (*dev-api.wrinkles.org*, *morbid-skinbags.wrinkles.org*, *420.shame.wrinkles.org*)and then fire off thousands of new subdomains at a DNS server, of which 99.9% of these will be lost into darkness. If it says that every ridiculous domain is valid, then we will have to parse it through a server with an option that can be told to test for and filter out crappy wildcard responses. OK, let's see what these tools can do for us.

Gotator, a tool that brags about being able to stuff over a million mutated subdomains within two seconds is written in Go (which is good) and is a staple in every recon session I've done for a while now. And also, so are the rest of these tools I'm going to show you, because they are all in a script together for maximum attack surface expansion madness.

`go install -v github.com/Josue87/gotator@latest`

gotator -sub creamsoft-subdomains.txt -perm permutations_list.txt -depth 1 -numbers 10 -mindup -adv -md > gotator-creamsoft.com

The one-liner above is pretty good, but here is what the arguments mean if you want to experiment with them yourself.

The **"-sub [subdomainsfile.txt]"** and **"-perm [permutationfile.txt]"** are your subdomain list and your permutation wordlist, in the formats explained at the beginning of this section. It is important to realize that since there are so many permutations in these tools for each subdomain, each subdomain and each line in your permutation list will increase the time the tool takes to run exponentially. When the output is too extreme for reality, I'll chop the subdomains down to ones that are resolved to a domain or swap the permutation file to a leaner one.

-depth [Number] If set to 1, the permutations would add one level of subdomains below your current list (as in api.doomed.com would be mutated to **dev.api.doomed.com**, **gentiles.api.doomed.com** but not **crt.goyim.api.doomed.com**, which would introduce a depth of **2**)

-numbers [number] This specifies how far up and down should numeric permutations go towards. For example, setting this parameter to **10, dev-2.whiteriders.net**, it will check **dev-3.whiteriders.net**, **dev-4.whiteriders.net**, up to 12. If there is no number attached to the subdomain, it will make 10 permutations, adding a number

-mindup This minimizes duplicates, beneficial for all sessions.

-md Uses input from your subdomains file and includes it in the permutations, so *dev.api.boiledalive.com* would churn out *dev-api.api.boiledalive.com, api.dev.api.boiledalive.com* and more.

-adv Adds a couple more permutation methods, including some switch-ups with the two text files and appending them with dashes.

Next, let's rip through a tool called **ripgen,** another tool that creates permutations. My simple one-liner for using this tool can be seen below.

```
cat badboy-com-subdomains.txt |  ripgen  | dnsx -t 1000
-silent -o ripgen-results.txt
```

Goaltdns, an update of the pythonic-based *altdns* tool from eons past, gives us another way to whip and flip our subdomains for esoteric victories.

```
go install github.com/subfinder/goaltdns@latest
```

Resolving Permuted Files

Since a lot of these permutation files have massive output and I'm definitely not trying to keep a massive list of 5 million fake domains making my laptop heavier by stinking up my hard drive, you should

avoid even letting these touch down in a file, instead opting to pipe them directly into a DNS resolver and saving only the valid subdomains you discover.

For maximum effectiveness, you need to have an updated list of DNS servers, to make sure each of your mutants is being accurately resolved and you aren't missing a golden goose of vulnerabilities. One effective pathway to this destination is through using a tool called dnsvalidator by **vortexau**. Grab it and install that bad boy via the commands found below.

git clone https://github.com/vortexau/dnsvalidator.git

python3.11 setup.py install

You can use the massive list of thousands of nameservers found at https://public-dns.info/nameservers.txt as your input, the great lists kept current by Trickest or any other reputable source of your preference. You can run it against the URL of your file without even downloading it via the command below.

```
dnsvalidator -tL https://public-dns.info/nameservers.txt
-threads 100 -o resolvers.txt
```

Assuming you already have a big, stinky and bloated file of fake subdomains, you can use the hyper-fast **puredns** by d3mondev to start cracking them open.

Now you can drill through these bad boys, save the valid ones it spits out and delete all these big, nasty files that serve you no purpose anymore.

```
puredns resolve stinky-mutant-subdomains.txt -r
resolvers.txt --write valid_domains.txt --write-wildcards
wildcards.txt
```

You can also pipe the permutation tool's output directly into puredns, so all those hideous subdomains never need to touch your hard drive.

```
gotator -sub targetsubs.txt -perm perms.txt -adv -md |
    puredns resolve | tee gotated-n-resolved.txt
```

The tool is also quite efficient with bruteforcing subdomains.

```
puredns bruteforce commonspeak.txt redrooms.com
```

Another tool fulfilling the role of insane bastard resolver, is projectdiscovery's **dnsx**.

You can install it through go.

```
go install -v
github.com/projectdiscovery/dnsx/cmd/dnsx@latest
```

Or grab the compiled release.

https://github.com/projectdiscovery/dnsx/releases/

I usually set it to record any response it receives like so.

```
cat badboy-com-subdomains.txt |  ripgen  | dnsx -t 1000
-silent -o ripgen-results.txt
```

If your output is popping regardless of the nonsense that is entered as a subdomain, you'll want to have dnsx use its very bleeding-edge wildcard filter. Apparently, it does multi-level DNS-based wildcards, which I am not afraid to admit is a term that I'm unfamiliar with. Add it by tossing in the "**-wd [domain.com]**" into dnsx's arguments.

Finally, you can knock out both brute forcing domains and permuting at the same time, you can check out bp0lr's tool **dmut**.

You know the drill at this point.

```
go install github.com/bp0lr/dmut@latest
```

My command of choice that I have saved to an alias is the following (your domain goes after the **-u** parameter).

```
dmut -u $1 -d ~/wordlist/dmut-permutations.txt -w 100
--dns-timeout 300 --dns-retries 5 --dns-errorLimit 25
--show-stats -o dmut-$1-results.txt
```

Bp0lr strongly emphasizes the importance of having a frequently updated list of resolvers, otherwise you're going to be missing

subdomains. You can use this built-in feature to invoke a fresh rotation of them that saves to **`~/.dmut/resolvers.txt`** via the following command:

```
dmut --update-dnslist
```

If something goes wrong for whatever reason, you can always access the files it pulls here: https://github.com/bp0lr/dmut-resolvers

And if those ever stop their continuous rotation of fresh resolvers, this seems like a pretty solid bet for the time being: https://github.com/trickest/resolvers

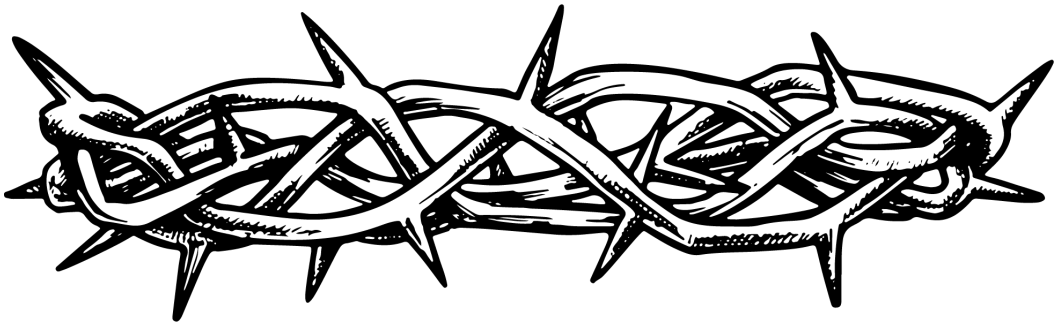
Subdomains acquired and ready to hunt?

Oftentimes, our excellent and comprehensive efforts have left us with a massive list of subdomains and a consequently herculean task of deciding what to investigate and attack. I almost always send my list through a custom httpx script I wrote that saves an output with as much data about each valid subdomain, in addition to a file of all the 200s, another with 404s, one with 403s and a list of all the IP addresses found. Then you can easily send the 403s list to a "403 Bypass" tool, start fuzzing the 404s for endpoints with a ffuf for-loop (because they probably have *something* on them) and the 200s to a spider like Katana, Hakrawler or GoSpider. You could also send everything to a screenshot tool (or just the 200s if your target is Yahoo or something) and pick what looks interesting visually.

If hunters were crazed hammerhead sharks, these signs would be akin to a cruise ship full of amputees with puncture wounds that just

capsized. Essentially, you will be more likely to find an easy meal with less than typical effort. Some strong signifiers include visually dated web sites (you'll know when you see it, the lack of web fonts, color palettes that make your tummy hurt , CSS only capable of creating rectangles everywhere and the corners on text boxes are so sharp your screen's pixels unalive themselves trying to render them), ancient messages found in the site's footers ("All Rights Reserved 2002") or outdated authenticated methods (any 401 status codes). Old frameworks are likely not updated and may have CVEs that have been festering in their bones for many years. If you see functionality that has many opportunities for lots of tests, such as registration, password reset, logins, file uploads and an abundance of customizable content, that subdomain elevates itself on the priorities list. After many successful years of hunting, zseano likes to target very large organizations that likely have many teams, oftentimes in internationally different locations. If these diverse teams are writing code in separate places, at separate times and there is some functionality that merges these two code bases, the lack of communication and incongruent work practices will likely produce pieces that don't fit perfectly together.

(To my USA friends: Do you still think that diversity is a strength?)



Content Discovery

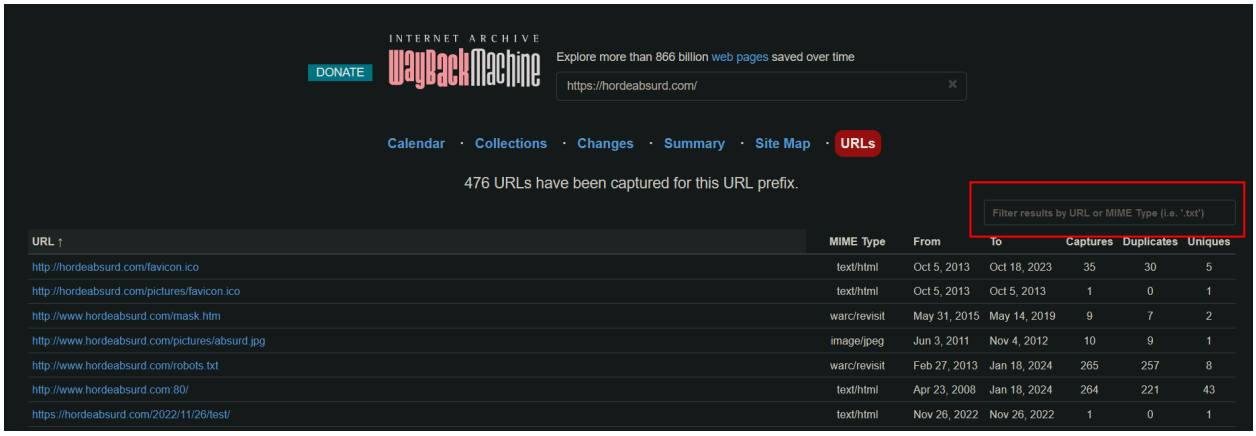
Once we've done all of our horizontal (domains) and vertical (subdomains) enumeration, we want to start finding active endpoints, pages or content that the server is hosting. This consists of static files that are default to the framework the page is using (boring) and customized content developed for the target domain (potentially not boring). There are a few primary ways to find these endpoints, consisting of...

1. Third-Party Endpoint Collection Services: consisting of querying websites like archive.org (Wayback Machine), urlscan.io, commoncrawl.org, alienvault.org and more
2. Spidering the website with a tool or proxy
3. Extracting endpoints from JavaScript files
4. Fuzzing endpoints with wordlists

Third-Party Endpoint Collection Services

For whatever purpose, these benevolent actors spy on everyone all the time and release the data to anyone that wants to see it. They are usually eternally growing, with some even storing the same page over and over across arbitrary time periods, like archive.org.

Archive.org, or the Wayback Machine, provides multiple versions of any page on the internet, as long as they spider it themselves or an individual requests it. There is also a useful way to look at the entire list of endpoints under a domain, via the URLs tab. Furthermore, you can filter the list for interesting extensions that commonly include dynamic functions (non-static pages), such as `.php`, `.aspx`, `.jsp`. You could also look for admin sections (`/admin/`), configuration pages (`.yaml`, `.json`, `.xml`, `.conf`) or JavaScript pages of yesteryear.



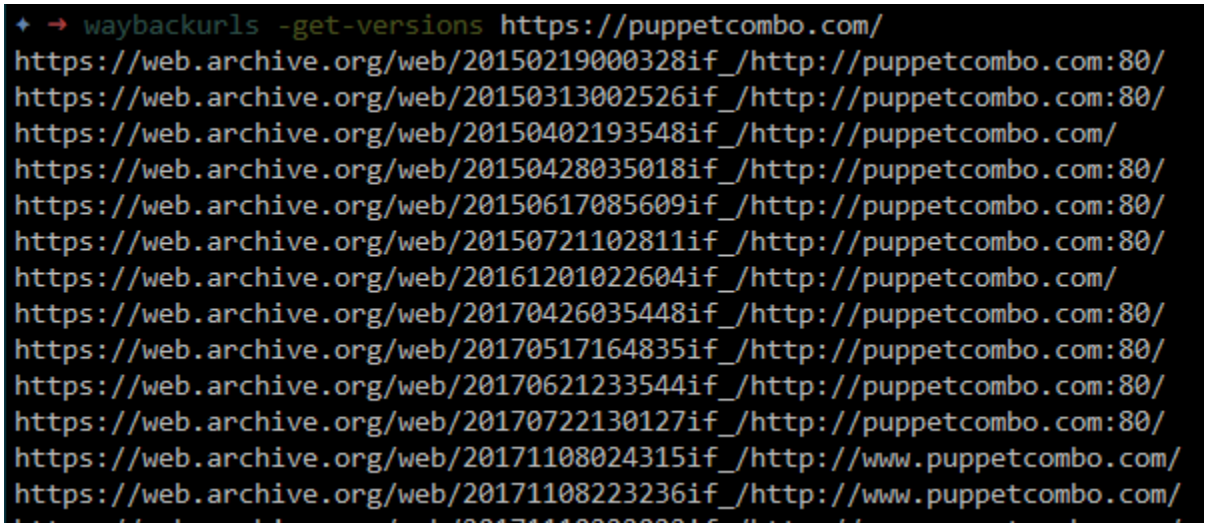
Besides being engaged through the web browser, it is a bit more efficient to use command line tools to pull much of this information. The most well-known being tomnomnom's **waybackurls**, which can be installed via slapping this into your terminal:

```
go install github.com/tomnomnom/waybackurls@latest
```

Usage is quite simple and ready for pipe travel, just throw whatever domains you want at it.

```
cat domains.txt | waybackurls > urls
```

A couple more options are available and can definitely be useful, including **-get-versions**, which pulls all available versions of a specific URL and saves them locally.



From the list in the image above, you can use **wget** or **curl** to pull any or all versions. A quick bash loop to accomplish this could be:

```
for x in $(waybackurls -get-versions
https://puppetcombo.com); do wget $x ; done
```

Or you could pipe it into another of tomnomnom's tools **concurl**. You can grab a release here: https://github.com/tomnomnom/concurl/releases

Essentially, you'd just pipe the output right into concurl and be done with it (the **-c** flag means concurrent download streams, which

shouldn't be set too high in this situation, as archive.org does give out some short IP blocks, though they are fairly generous with their server limits)

waybackurls -get-versions https://puppetcombo.com | concurl -c 3

A more recent and more thorough approach to scraping this poor service can be found in xnl-h4ck3r's tool called **waymore.** This tool truly does deliver way more results than anything else out there, though it does take a bit more time (but you have plenty) and a substantial bit more bandwidth (but you are using a cloud machine so who cares). Instead of just pulling the URL list from the Wayback Machine, it can pull all of the archived pages from all times, so you can parse their contents for more links, developer comments, secrets and whatever else lay dormant waiting. Some tools that pull results from third-party URL aggregating services hit rate limits and wrap up their business, without telling you the results may be incomplete. Waymore is aware and will wait until rate limiting cools down a bit, which also takes waymore time but I think it's worth it. Just make sure you automate it and take a nap, instead of drooling at your desk and you won't even notice the time pass. Waymore queries CommonCrawl, VirusTotal, Alienvault and Urlscan, so I don't even really need to talk about any tools that query those services on their own.

git clone https://github.com/xnl-h4ck3r/waymore.git

python3.11 setup.py install

To pull just URLs (otherwise, the default will pull URLs and responses):

`python3.11 waymore.py -mode U -i domain.com`

The URLs will be automatically saved into the folder that waymore was download to, under **`./results/domain.com/.`** It's important to be aware of how much content your target has as it can be an unreasonable amount of hammering against the API providers and your own machine (for example, Twitter.com would pull 28 million pages of content). You can remedy this situation and come out with something that doesn't put you on suicide watch a few different ways.

- You can use **-l [integer]** to limit the amount of responses that are saved. The default is 5,000, while 0 means unlimited (as in, all of them). A positive number of 500 would download the first 500 results, while entering -500 would download the 500 most recent results.
- You can use the **-ci [h/d/m]** flag for designating preferred capture intervals to hourly, daily or monthly. For example, if set to monthly, it would only download one response per month, regardless of how many versions are saved on archive.org.
- You can use **-mc [status-code]** to only download responses that match a certain status code, i.e. 200 for valid responses.
- You can use the **-ko** flag, followed by a regex value that the response must match in order to be stored. For example, to pull only the JS files, you would type **`"-ko '\.js(\?|$)'"`** as an argument.

There's even more ways to limit the potential avalanche of results this tool can suffocate you with, such as limiting the CommonCrawl databases it will parse, setting limits on beginning and end dates (no data before 1997 will likely still be available on any server in its unchanged form) and more. It's an incredibly valuable and deep tool and I highly suggest you check out and alter the **`config.yml`** file that comes with it to really customize the data output you're looking for.

Ok, there is one place that **waymore** doesn't pull URLs from and that is across all Github repos. Fortunately, there's an app for that, gwen001's **github-endpoints**. As with his subdomain extracting tool, set-up some Github API tokens into a file, one-per-line and you are good to go.

Install via golang in your terminal:

```
go install github.com/gwen001/github-endpoints@latest
```

Then run the tool like so:

```
github-endpoints -t ~/github-tokens.txt -d domain.com
```

Also, this tool is great when you need a super fast list of URLs and has been a staple in my arsenal so it would fill me with a feeling of incompleteness if I didn't mention it; **gau** by lc.

```
go install github.com/lc/gau/v2/cmd/gau@latest
```

The syntax is simple, though I always include a couple flags to filter crap out of my results.

```
gau -t 200 -subs  -b
ttf,woff,svg,png,jpeg,jpg,gif,ico,woff2,css -random-agent
domain.com
```

The **-t** flag for threads, **-subs** for including subdomains, **-random-agent** to alleviate the likeliness of being IP banned just a pinch and **-b** for blacklist, followed by a bunch of extensions I truly hate seeing in my results.

A Quick Notice to Users of Oh-My-Zsh Framework

Years ago, when I was far more confused about what I was doing in Linux, I had a long battle that took longer to resolve than I can admit without shame. Oh-My-Zsh comes with a Git plugin enabled by default, which contains an alias for **gau** (git add --update) that conflicts with this URL-scraping tool. You can either rename the **gau** binary (to **getallurls** or whatever) or edit the git-plugin file and comment out the offending line (found at **~/.oh-my-zsh/plugins/git/git.plugin.zsh**)

Another tool that hits a couple more third-party services is hueristiq's **xurlfind3r**. Often grabbing me a handful more URLs to add to my aggregate list, this tool also hits services like intelx.io (steal some keys) and bevigil (free API keys). Make sure to fill out the config file with the API keys though or all of this is meaningless.

Install it via:

```
go install -v
github.com/hueristiq/xurlfind3r/cmd/xurlfind3r@latest
```

This tool also has the ability to take a list of domains as an input (**-l list.txt)** , can parse a domain's robots.txt files from the Wayback Machine (**--parse-wayback-robots**) and even the source files of past versions (**--parse-wayback-source**). You can also filter by regex, for example, to pull only JavaScript files from the output.

```
xurlfind3r -d domain.com --include-subdomains
-m '^https?://[^/]*?/.*\.js(\?[^\s]*)?$'
```

Finally, another tool I really use consistently when I want a quick overview of a domain's parameter entrypoints so I can get right to punching in some injection payloads (yes, the "spray and pray" or "low hanging fruit" method) is devanshbatham's **paramspider**. It works by pulling all the endpoints it can find across multiple third-party services but only if they have parameters (so you can just go right for the XSS, SSRF, LFI, SQLI type vulnerabilities immediately without any fat getting in the way).

git clone https://github.com/devanshbatham/paramspider

You can use **-d** for a single domain or **-l** for a list of domains.

paramspider -d boltthrower.com

Your output is automatically saved and parameter values are filled with "FUZZ" as a default. If you want to swap the value for something else use the **-p [FUZZ-value alternate]** syntax. A potential use could be dropping in a XSS payload and then piping it to httpx with a matching

response regex to only see the URLs that reflect your payload for your output.

paramspider -d 20buckspin.com -p %00%00%00%00%00%00<script>alert(313313313)</script> | httpx -silent -mr 313313313

Those few tools combined should get you a sweet set of scraped URLs to start the endpoint enumeration game with. Combing through online resources is only part of it and there's plenty of hunters that just run "`waybackurls domain.com`" and nothing else. That is not the way.

To wrap up this mini-section, here's some one-liners to query these services yourself in case you're making your own automation tools and don't want to rely on anyone else.

Alienvault.com:

```
curl -s "https://otx.alienvault.com/api/v1/indicators/domain/domain.com/url_list?limit=100&page=1" | grep -o '"hostname": *"[^"]*' | sed 's/"hostname": "//' | sort -u
```

Archive.org:

```
curl -s "http://web.archive.org/cdx/search/cdx?url=*.domain.com/*&output=text&fl=original&collapse=urlkey" | sed -e 's_https*://__' -e "s/\/.*//" | sort -u
```

Commoncrawl:

```
echo "domain.com" | xargs -I domain curl -s "http://index.commoncrawl.org/CC-MAIN-2018-22-index?url=*.domain&output=json" | jq -r .url | sort -u
```

Spidering For Endpoints

Part two of the endpoint expansion saga will discuss the well-known art of spidering. Spidering works in much of the way that search engine bots index the internet; you feed it a page and the spider extracts all the links from it. Then each of those links is visited and the subsequent page is scanned for links. You repeat this until you discover the entire internet or run out of new places to look. This is generally done automatically, though doing some of it manually can give you a much better understanding of the functionality and organization of the web app you are testing.

For very basic smooth brain manual spidering, I like to use a browser add-on called “**Link Gopher**” that spits out a swath of links inherent to your current page.

Chrome: https://chromewebstore.google.com/detail/link-gopher/bpjdkodgnbfalgghnbeggfbfjpcfamkf?pli=1

Firefox: https://addons.mozilla.org/en-US/firefox/addon/link-gopher/

Output looks like (plus a list of domains discovered as a separate list on the bottom):

Links

https://store.eisenton.com/
https://store.eisenton.de/afsky-a-2863
https://store.eisenton.de/agalloch-a-426
https://store.eisenton.de/cantique-lepreux-a-2710
https://store.eisenton.de/de/
https://store.eisenton.de/en/
https://store.eisenton.de/en/#
https://store.eisenton.de/en/#box-slides
https://store.eisenton.de/en/campaigns
https://store.eisenton.de/en/cancellation-withdrawal-digital-goods-i-9
https://store.eisenton.de/en/cancellation-withdrawal-i-4
https://store.eisenton.de/en/checkout
https://store.eisenton.de/en/create_account

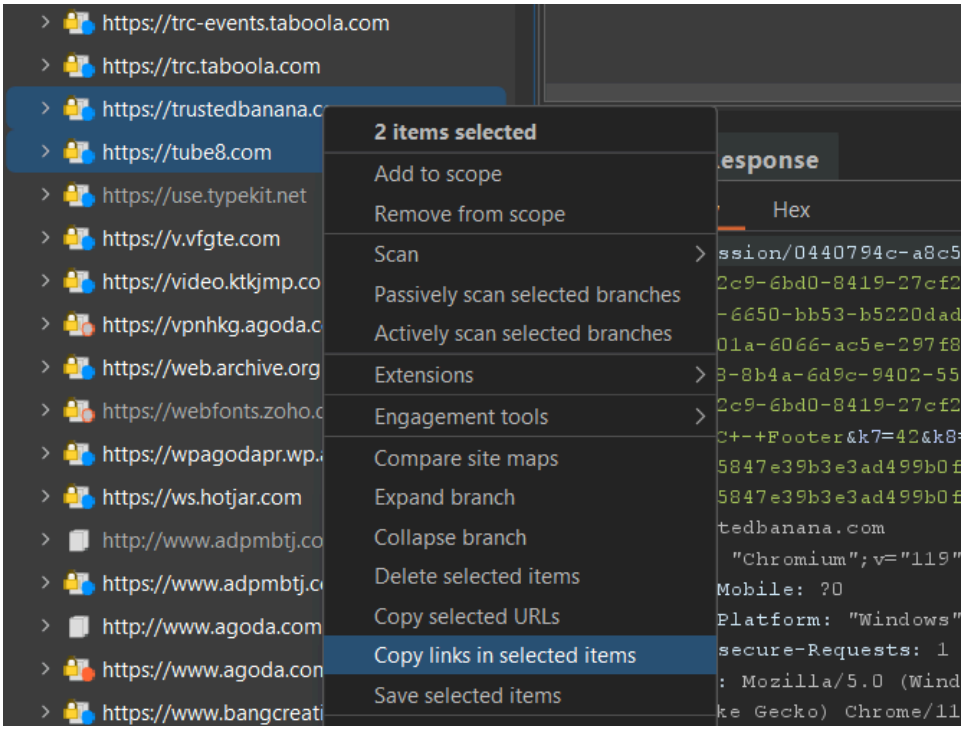
Sad Man's Browser In Solitude Aeternus Method

In your browser, open the Dev Tools panel (right click + "Inspect" or F12, generally) and go to the "**Network**" tab. Spider the whole site yourself, staying within one tab and don't close it (!!). At the end of your long and sad session, highlight all the requests you made to the many resources across the internet you pulled from, then right click and choose "**Save All as HAR file**". Finally, your last step is to do your own research and become a pioneer - since you are now the first person to ever actually try this method. My tactical search engine OSINT has allowed me to grace you with a couple mystic hints… hmmm… looks like… "HAR-READER" … or maybe… "HARLIB"? This stuff sounds crazy, buddy, you're on your own with this one!

The next method combines a manual approach with a tool-based one, though it requires Burp Suite Professional Edition, which is worth it if you're planning on being a lifer in this field. Jason Haddix is the only

person I've heard really go into this so I'll just briefly touch upon it so you can catch a buzz and go to the source for the full dose.

1. Turn on Burp Suite, open up your proxied browser and collect all the requests and whatever they happen to suck up along with them. (Don't discriminate or scope gatekeep!)
2. Turn off passive scanning, we're just sucking up buttloads of endpoints here. For scope, turn on advanced mode (regex) and just use the keyword to filter endpoints (for example, just do meatballs instead of meatballs.io).
3. Spider yourself as much as you can across your target domain and then spider all the wonky hosts you picked up (with the spider feature, not manually this time). You can also highlight all the hosts that are in scope and send them to one of the spidering tools I'm mentioning in a couple shakes of a lamb's tail (katana, gospider, hakrawler).
4. You can also highlight anything that looks relevant and right click to "**`Copy Links in Selected Items`**".



To export all this data, you can hit "**Engagement Tools"** > "**Analyze Target**" or save the output as an HTML file. Honestly, there isn't a super clean way to interpolate the data from this into your current methodology or plans for automation but you can end up with a ton of extra places to look, which is awesome if you have a huge scope.

Now, let's get into spidering with tools. There have been a couple classic spiders for years but recently, projectdiscovery's **katana** has been my go-to champion that was immediately integrated into my scripts. Here she is, fellas.

```
go install
github.com/projectdiscovery/katana/cmd/katana@latest
```

You can obviously run it like a stupid idiot, as seen below.

```
katana -u https://moronbrothers.org
```

But I hope you are far and above this type of behavior by this stage of the book. If not, I guess you can just live in the hell that you created for yourself and when you're ready to pull yourself out of it, do that and send me a message to let me know how you did it.

My general-use katana command looks like the following:

```
katana -nc -jc -kf -fx -xhr -aff -jsl -ef
woff,css,png,svg,jpg,ico,otf,ttf,woff2,jpeg,gif,svg -r
~/resolvers.txt -list my-urls.txt -o katana-output.txt
```

A quick explanation is due, I suppose.

- **-nc** : no colors, sometimes they mess with the output when piping to other tools and put hideous artifacts like "[92m" at the beginning of each line.
- **-jc** : enables parsing discovered JavaScript files and crawling their endpoints
- **-kf** : enables parsing and pulling from "known files", such as sitemaps and robots.txt, then feeding them back into the input
- **-fx** : enables extraction of form, input, textarea & select elements and feeds those values into the input to discover more endpoints
- **-xhr** : extracts XHR parsing magic during headless mode for some more potential discoveries
- **-aff** : automatic form filling, to potentially discover new endpoints via the web app's logic of processing requests

- **-jsl** : utilizes the magic genius of JS parsing with jsluice, a tool I will discuss in a little while
- **-ef :** removes hideous extensions that pollute your URL list
- **-r :** being my own resolver list
- **-list [your URLs.txt]** or replace this with **-u https://sweetbaby.io**

You can also set its spidering strategy, tighten and loosen scopes and some more wild features. This tool is still pretty fresh so I'm sure it will have consistent additions of new features but this is a very righteous origin story. There's other great tools in this lane as well, like **Gospider** by jaeles-project and **hakrawler** by hakluke, both of which I used extensively for years and had no issues with. After doing some test runs against katana, the results were fairly close but katana edged ahead. I'm going to assume it's because of the features that other crawlers don't have, like integrating the Javascript parsing capabilities of **jsluice** to conjure and feed itself new potential endpoints, the experimental form-filling capabilities and the flexible headless crawling that hasn't be challenged or even imitated in the current offensive web crawlers landscape. The ingenuity and execution of some of you human beings never ceases to amaze me. Well, we live in a society.

Endpoints from JavaScript Files

Part I: Separate and Stockpile

As we descend through the endpoint gathering rituals performed by hunters across the ages, this one may be the most revered. Javascript files are often filled with hardcoded API keys, secrets, credentials and other quick slam dunks, but you have to parse carefully. Since a lot of the value here obscures itself from automation, there is generally stuff to find, since lots of people don't want to do the work. Beyond hardcoded treasures, these files describe other endpoints, HTTP methods, some semblance of fragmented and primitive API documentation. These files often lead to all kinds of bugs and endpoints, as well as being prone to subtle changes over time that often introduce their own vulnerabilities. There's a bunch of different ways to attack this topic so I'll just start ***hacking*** away at it.

To start off this frenzied JavaScript methodology, my first step would be to gather all of the endpoints from the past steps in this section - the ones pulled from various third-party services and then spidered to find as many relevant endpoints as possible. At this time, you could use

something like **httpx** or **fff** to weed out all of the bad endpoints (**cat urls-gathered.txt | httpx -mc 200,302**), though I think **katana** should've done all of that if you feed the third-party lists to it as your input. Once you have this list, you can pull all the JS files from it with the following command.

cat urls.txt | grep -E -v "\.json|\.jsp" | grep ".js" | tee js-urls.txt

To make sure that none slipped through the cracks of reality or ended up somewhere chaotic, I like to run my URL guys through a tool called **getJS** by 003random. Install away, my friends.

go install github.com/003random/getJS@latest

You feed it your URLs and it will extract all the JavaScript files within them for you.

cat urls.txt | getJS --complete --resolve --insecure | anew js-urls.txt

The **--complete** flag will make the tool print full URLs rather than the paths, **--resolve** ensures that only valid, accessible endpoints are output and **--insecure** will allow the tool to work in spite of some potentially grotesque certificate upkeep habits on your target's side. I'd also recommend that you don't be too exclusionary at any stage in the JS enumeration segment with cutting out URLs that don't appear to be in scope. Many times, organizations host JavaScript files on CDNs, cloud hosting services, Google Analytics and more, yet they are still unique files specific to your target, rather than a generic JS library file that is shared across the entire internet (there are those also, and they will

bloat your data and waste your time, but you cannot sacrifice some good boys just because of this).

Alternatively, if you've been exploring the web application through Burp (and you have the Pro version), you can highlight all your target URLs, right click and go to "**Find Scripts**" - which grants you the ability to save them all into a big, sad file that you can grep through, I guess. Not my favorite method but it exists - at a cost. 🙂 During dark times like these, heroes arise through the fog of anti-cosmic confusion and create their own rules for the rest of us to follow. One such benevolent force of lawful neutrality is known as 0xDexter0us for providing the tool **uproot-JS** (grab a release at https://github.com/0xDexter0us/uproot-JS/releases). Drop it as an extension into Burp Suite and then you can find a new option to save the JS files in the top menu.

Endpoints from JavaScript Files

Part II: Extract and Extend

After spending considerable time and effort into finding and separating our JavaScript files, let's get to choppin'. This is an area where the more time you spend staring at these walls of (often hideous) code, the more rewarded you will be. It's OK if you don't want to do that, because there's another path with less rewards, however, it is presented with less required suffering. You can use tools to pull the obvious things out, but it's likely a handful of talented young hunters have already spun the

block on these endpoints before you even got invited to the program. Let's break down some methods for parsing these guys.

1. *Manual human eyeball scanning* and collaboration with the human brain to understand data through cold, slow reading. If you can't do this at all, then this is the worst solution for you. I recommend you find a fun way to learn basic Javascript (reading will be enough, though being able to write will definitely help you) and Gareth Hayes' book on **"JavaScript for Hackers"** to work on this shortcoming, but you can definitely cushion your methodology with tools that will literally highlight things that might be interesting. But no tool is able to cleverly decipher how these files could all be related and predict potentially interesting but unexpected strings, so a manual approach utilizing our unique cognitive abilities will always be preferred here.
2. Feed all your JavaScript files into tools that scan their contents for interesting strings, such as hardcoded API keys or credentials, or common functions that are often ripe for exploitation. It is likely that the source code holds hidden parameters and endpoints that the spidering tools didn't catch, potentially due to their uncommon presentation or even obfuscation to intentionally make their discovery less likely. Some neat tools to use for this include tomnomnom's **gf**, **nuclei**'s Javascript-focused templates and being clever or creative with **grep**.
3. You should also use tools that parse Javascript files and find links for you, that you can then turn into a wordless to fuzz your target domain (and its associated domains and even completely unassociated domains).

Since there's some overlap between the second and third methods, I'll proceed to tell you what to look for and which tools can help you find the secrets and the expanded attack surface together. In our Javascript files, we want to discover...

- More *endpoints*. You can grep for basic slashes **"/"**, protocols **"https://"** or parts of paths like **"/api"** or **"/users"**. API calls will often be connected to HTTP methods (**"POST", "GET"** or simply the word **"method"**) or connected to strings like **"send("** and **"api"**. Endpoints sometimes mention the headers required in the request for a successful response, so you can also grep for strings like **"Content-Type"**, **"setRequestHeader"** and **".headers"**. Finally, searching the JS file for your target's domain name can point you towards hidden subdomains and endpoints too.
- *Parameters*. Javascript files will often describe how an endpoint should be interacted with and what parameters are needed for a successful API call. Grep for strings like **"parameter"** and **"getParameter()"**, or even specific parameters connected to vulnerabilities, like **"return="**, **"redirect"** or **"url="**. You can also add "**var "** and then test the name of the variable against any relevant endpoints with various injection vulnerabilities.
- *Leaky treasures* such as API keys, tokens, secrets and credentials. You can use various tools or grep for these yourself with terms like **"secret", "admin", "token", "passw", "debug"** and more.

Oftentimes, your target's Javascript files will hideously unreadable, intentionally done so via obfuscation, minification or uglification. You can try to reverse this process through various sites like https://beautifier.io/, https://beautifytools.com/javascript-beautifier.php or potentially stacking multiple methods.

My most successful method has been using **js-beautifer,** which can be installed like so:

npm install js-beautify

(Install nodejs and npm through your distro's apt-get equivalent)

Let's get into the bleeding edge of Javascript illumination methods in a very rapid fire way.

Getting URL endpoints from JS Files:

- First, a simple one-liner that will pull the very obvious endpoints from your JS file.

cat file.js | grep -aoP "(?<=(\"|\'|\`))\/[a-zA-Z0-9_?&=\/\-\#\.]*(?=(\"|\'|\`))"

- And another, by sratarun via Twitter, for pulling all parameters from a JS file and then checking if they are valid on their respective endpoint.

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

Of course, you can replace assetfinder and gau with "**cat urls.txt**" or something similar if you already have a good list to work with.

- **Jsfinder** by kacakb. (https://github.com/kacakb/jsfinder)

Install via:

go install -v github.com/kacakb/jsfinder@latest

It extracts endpoints via Javascript tags like "src", "href", "data-main" and more. You can use it via piping a list of URLs like so:

cat list.txt| jsfinder -read -s -o js.txt

Or via command line arguments as below:

jsfinder -l list.txt -c 50 -s -o js.txt

(**-c** is concurrency, which we can push above the default of 20 since we are all using cloud machines, **-l** is your list and **-o** is output as expected)

- **Aranea** by leddcode (https://github.com/leddcode/Aranea)

A unique approach that does some neat analysis across a single JS

file or the tool is also able to crawl your target as well.
Clone the repo and basic usage for a single JS file can be seen below:

python3 aranea.py -U https://example.com/script.js -M analysis

For the crawling features, use the *crawl* method instead of *analysis*.

python3 aranea.py -U https://example.com -M crawl -T 100

Without pointing it at a specific JS file, it will attempt to find *main.js* or its equivalent. If your JS file is rotten and corrupt inside and avoids using a *.js* extension, add a **-S** flag (probably for shameful, sad or stubborn).

- The golang version of the classic linkfinder, **GoLinkfinder** by 0xsha (https://github.com/0xsha/GoLinkFinder) is probably the most well-known tool for this purpose.

go install github.com/0xsha/GoLinkFinder@latest

Usage is simply inputting a URL (though deceivingly with the **-d** parameter, generally used for domains).

Golinkfinder -d https://invictusproductions.net/ | tee golinkfinder.txt

- **EndExt** by SirBugs is one I really like for its purity and simplicity. Clone the repo and then build the go script into an executable file.

go build main.go

mv main /usr/local/bin/endext

Now you can run it from anywhere via:

endext -l js_files_urls.txt

You just feed it a list of JS endpoints and it sends out a list of nicely formatted links

- **xnLinkFinder** by xnl-h4ck3r (**https://github.com/xnl-h4ck3r/xnLinkFinder**)

Probably the most intense tool for finding URLs and has the fattest output as well. Clone the repo and install via python3. The most basic usage is as follows:

python3 xnLinkFinder.py -i target_js.txt -sf target.com

Both the **-i** flag (followed by a list of JS URLs) and **-sf** flag (followed by in-scope domain) are required.

You can also insert a Burp Suite save file.

python3 xnLinkFinder.py -i target_burp.xml -o target_burp.txt -sp https://www.target.com -sf target.* -ow -spo -inc -vv

The **-sp** flag is for scope prefix, to remove all the trash that browser proxies absorb from all over the internet and **-inc** will include the input links in the output as well.

My favorite features of this tool include the ability to parse endpoints into multiple outputs, including a list of parameters discovered and a general wordlist for fuzzing endpoints, parameters, subdomains or anything else you can think of using it for. Peep the syntax below, my friend.

python3 xnLinkFinder.py -i urls.txt -o output.txt -op params.txt -owl wordlist.txt -sf voidwanderer.com

- **Jsluice** by bishopfox

This tool appears to have made a quick rise in many bug hunter's toolkits and from a look at its impressive code, it is pretty deep, versatile and quite impressive at catching weird edge cases or unique JS attributes and parsing them successfully.
Installation is done through go:

go install github.com/BishopFox/jsluice/cmd/jsluice@latest

It has multiple modes but the most useful for me is the URL extraction one. Syntax can be seen below:

cat urls.txt | jsluice urls | tee jsluice-urls.txt

There is a lot more information you can parse through via the official Github page (https://github.com/BishopFox/jsluice), as well as a series of blog posts beginning on the BishopFox blog.
https://bishopfox.com/blog/jsluice-javascript-technical-deep-dive

Finding sensitive and interesting things in JS files:

- **jsleak** by byt3hx

A quick tool to easily insert into your automation is the simple and effective jsleak, install as below:

go install github.com/channyein1337/jsleak@latest

It can be used to pull links via **-l** (extract the superior full URLs with **-e** instead) and secrets via **-s**.

cat urls.txt | jsleak -s
cat urls.txt | jsleak -e

- **jsluice** by bishopfox, mentioned above can also be used to look for secrets.

cat urls.txt | jsluice secrets | tee jsluice-secrets.txt

- **SecretFinder** by m4ll0k (https://github.com/m4ll0k/SecretFinder)

Clone the repo and install the requirements with pip.
SecretFinder will run through a whole domain looking for goods.

```
python3 SecretFinder.py -i https://example.com/ -e
```

Ignoring common JS files that exist unchanged across various CMS and frameworks can be ignored via the **-g** flag. You can add multiple values into (just one) parameter by using a semicolon to

separate them as seen below:

-g 'jquery;bootstrap;api.google.com;woocommerce;_next'

- **Carridi** by edoardott (https://github.com/edoardott/cariddi)

Install via:

go install -v

github.com/edoardottt/cariddi/cmd/cariddi@latest

I like to max out the tool to its ultimate effectiveness, which can make a large domain take quite a while, as it crawls, parses and crunches endlessly. The key to ultimate power with cariddi is the following:

cat urls | cariddi -oh target_name -rua -e -s -c 200 -err -info

The **-oh** flag saves the output as a nice HTML file, **-rua** uses a random user agent to avoid being banned by some servers, **-e** for juicy endpoints, **-s** for secrets, **-err** for hunting for error messages, **-info** for interesting information and **-c** for concurrency, set to a vindictive and cruel level of 200 simultaneous palm strikes to the target server.

- **JSFScan.sh** by KathanP19 is one of my favorites.

Clone the repo and set-up through Docker for best results. I have

set it up myself by altering the script and it was still messy (due to my failures, not KathanP19's). While in the repo's folder, enter:

docker build . -t jsfscan

docker run -it jsfscan "/bin/bash"

And a pre-configured session should begin. Have your target list as a list of full URLs (beginning with http and all that). Usage is pretty easy beyond that, as I like to run the shell script (**./jsfscan.sh**)with the **-all** flag, as it gathers links, endpoints, secrets, makes a wordlist, hunts for DomXSS possibilities and puts it all together in a nice HTML report at the end.

./jsfscan.sh -all -r -o -l target.txt

The **-r** and **-o** flags create the report and include all findings in an automatically generated output directory, while **-l** points at your target list. This one does pretty much everything you would ever want your JS punching toolset to accomplish.

Since you made it this far, I'll toss you some info on a tool that can make you very lazy but I think you've earned it. **Scripthunter** by robre combines and automates multiple JS hunting tools and features into a simple and direct bash script for you to blast your targets with. Clone the repo at https://github.com/robre/scripthunter and make sure you have the required tools it needs to run (or not, you'll discover they are missing extremely quickly). This is a sweet tool that pulls JS files from a few third-party URL stockpilers, then does a bit of spidering, a bit of

fuzzing and some analysis. The fact it uses custom wordlists already goes beyond 90% of hunters and the way it absorbs data, digests it and creates more data to feed itself with is reminiscent of my spirit animal, the ouroboros, so I am naturally drawn to its magnetic energy.

Clone the repo and run the tool with this simple syntax:

./scripthunter.sh https://voidwanderer.com

One drawback to this tool (and all automated tools) is an issue that is known as the “Christmas Lights Conundrum”. With some cheaper brands of Christmas lights, when one bulb breaks, all the following bulbs decide to unalive themselves too. Similarly, automated methodologies and frameworks have the potential for something to be a tiny bit wonky with your target and it derails one portion of your chain, causing everything after it to wither and die. For example, the spidering portion might not know how to parse some weird technology on the first URL it tries, decides it's done and feeds an empty output to the next tool, continuing this sad chain of passing empty data all the way to your text editor where you excitedly awaited some interesting results. When you have the benefit of doing this work one bloated pipe-heavy terminal command at a time, you can catch these unique intricacies and adapt to them immediately, saving yourself from being a detective that follows the trail of empty text files backwards to where it all went wrong in your bash script.

> Don't forget to check your JS files for comments by developers that turn these source files into *sauce* files.
>
> (ᵔʖᵔ)

Before you report your "sensitive" secret...

You should check if the API key you found is still valid before you report a useless and expired (or worse, intentionally public and harmless) key to spare yourself time and humiliation. There's a great repo called **keyhacks** (https://github.com/streaak/keyhacks) by streaak, that is essentially a cheatsheet for tons of different API keys and checking their validity, typically with a quick curl request. There's also an interactive version here (https://github.com/gwen001/keyhacks.sh) by gwen001 that is nice to have ready to fire off in your arsenal.

If you find a Google Maps API key, I'd recommend you enter it into ozguralp's **gmapapiscanner**, which will tell you the various methods the key can be abused or not. Unfortunately, reporting this vulnerability is hit-or-miss depending on the program, I have seen people get decent bounties and I have also reported it, with a thicc report that clearly shows examples of me burning through $3 of credits in seconds but they didn't find it too worrying. There's been some changes to the terms, usage and settings in the past few years by Google, that has changed how some programs view the business impact of the bug. It can definitely be abused to waste money, burn through monthly credits and barricade customers from using the Google Maps functionality in a web app. This isn't too important if it's just the mini-map on the

"Contact Us" page but for a rideshare or food delivery application that incorporates it heavily in their workflow, a denial of service to the map integration could be tragic.

FROM SOURCE MAP TO HORSE CRAP

For speed and security, many web developers compress their Javascript, CSS and other asset files into hideous and frustrating blobs of characters; typically done through an uglify-ing tool called Terser. Though browsers can correctly render the mystifying code, a "source map" is generated throughout the uglification for debugging purposes (a few frameworks use a standard one though). This file provides a map that can return the compiled code back to the original code. These source maps typically end with extension ".map" (like *tragedy.js.map* or *ab.css.map*).

Elucidating these bad boys is now trivial through this feature in Chrome browsers and I've never seen any hunters mention it.

- In Google Chrome, go to "**Developer Tools**", go to "**Settings**" (the cog wheel icon) and at the bottom of the "**Sources**" column is a feature named "***Allow DevTools to load resources, such as source maps, from remote file paths***" which is disabled by default.
- Under "Experiments", there is also the potentially helpful feature, "***Resolve variable names in expressions using source maps***" and the potentially unstable and dangerous feature "***Use scope***

***information from source maps*”.**

- Now you can click the three vertical dots to the right of the cog icon, go to “**More Tools”** and choose “**Developer Resources”.** The newly spawned tab will show the location of any connected source map files.

A couple tools exist to assist you with this task. First, the browser based Source-Map-Visualization by sokra

https://sokra.github.io/source-map-visualization

and npm’s “**unmap”**, installed via **“npm -i unmap”**

To use unmap, download the .map file you are targeting first, then:

unmap ./app.js.map

The tool will output a list of URLs extracted from the source map, although it is slightly silly because they will be prepended with your current directory.

`””¯\_(ツ)_/¯`”

Fingerprinting : How To Label and Stereotype Your Enemies

This section will be fairly brief, as fingerprinting is rarely very difficult and since life is fair, the rewards for playing this game tend to align with the challenge it takes to get to the “prize”. Essentially, we want to figure out our target’s tech stack, which will allow us to more finely customize our strategy and adapt our methodology for further enumeration. This knowledge may hint at which vulnerabilities are more likely to be present or perhaps, in some unrealistic utopian pentesting fantasy, the version of the obscure CMS you are testing matches with a metasploit module released last month and put a plastic bag over your head to make sure you aren’t dreaming. Oh yeah, and it isn’t a CTF either. Maybe you keep looking for fingerprinting articles from time to time, thinking that there must be something deeper, some skill that you can sharpen to be better than someone else but maybe you’re just like me; and you keep searching and waiting.

Wappalyzer is a popular browser extension (and now Burp Suite extension and Golang library) that fingerprints the current site you are browsing and goes impressively deep, telling you the Javascript

frameworks, libraries, protocol versions and some things that would actually be difficult to fingerprint, surprisingly.

For example, did you know that Google Docs is written in Java?

That fact took my mind on a journey. So imagine this: since a young age, you worked really hard to learn programming. Making some 2D platformers, then making a bit of cash designing websites as a teenager and then started taking it very seriously. So seriously, that it became your life and eventually, you applied to Google. After an *OK* performance, you were kinda bummed that you probably wouldn't get the job. However, a couple weeks later, you receive a call. The called ID says "John Hammond"? Congrats, you made it!
You wear your nicest Moncler polo to your first day at work when you hear the words that immediately make you recoil and wonder how your life could have been different had you not wanted to understand computers.

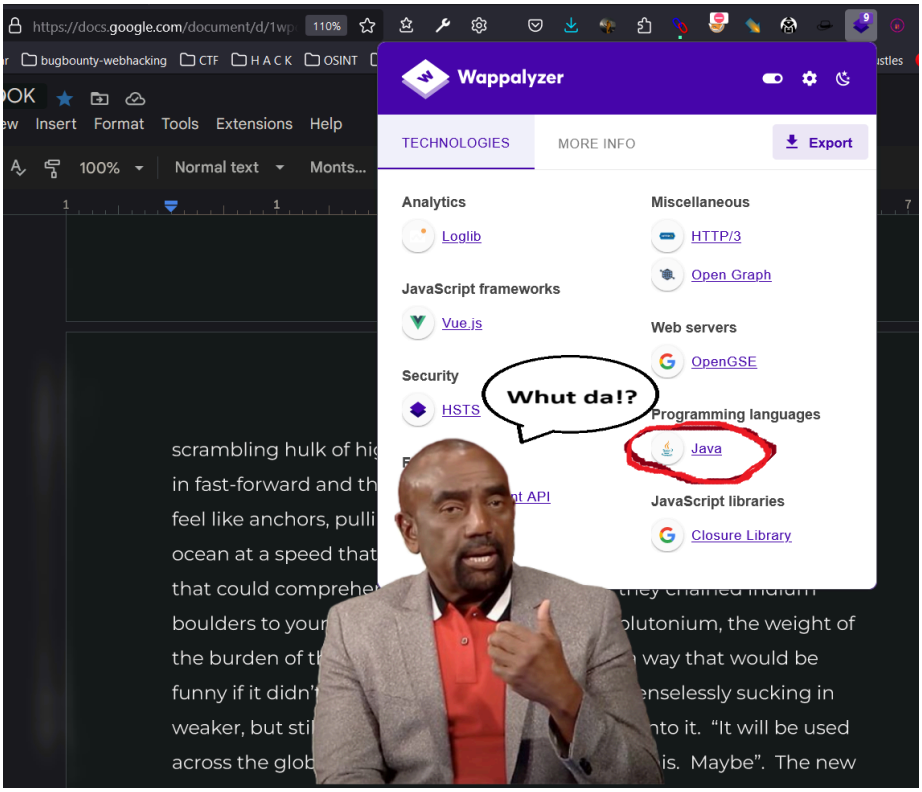
"You're gonna make Google Docs…. in Java!"
The last two words seem to tear your soul from your skeletal frame, repeating and bouncing in your head like ancient war drums beating to the death march of the foreign armies that will raze your family lineage to a violent end. The sound of those last two words is almost inhuman, as if the mid-tones were slowly scraped from the tonality and left a scrambling hulk of high pitched shrieks, like rusty guillotines wrestling in fast-forward and the opposite end, where the tones are so low they feel like anchors, pulling your withered husk into the bottom of the ocean at a speed that would terrify you; were you still in a human form

that could comprehend its surroundings. As if they chained iridium boulders to your back and coated your flesh in plutonium, the weight of the burden of the task would be incalculable in a way that would be funny if it didn't have the gravity of a black hole senselessly sucking in weaker, but still unfathomably dense nearby black holes into it.

"**Maybe. It will be used across the globe, possibly to write stupid shit like this. Maybe**".

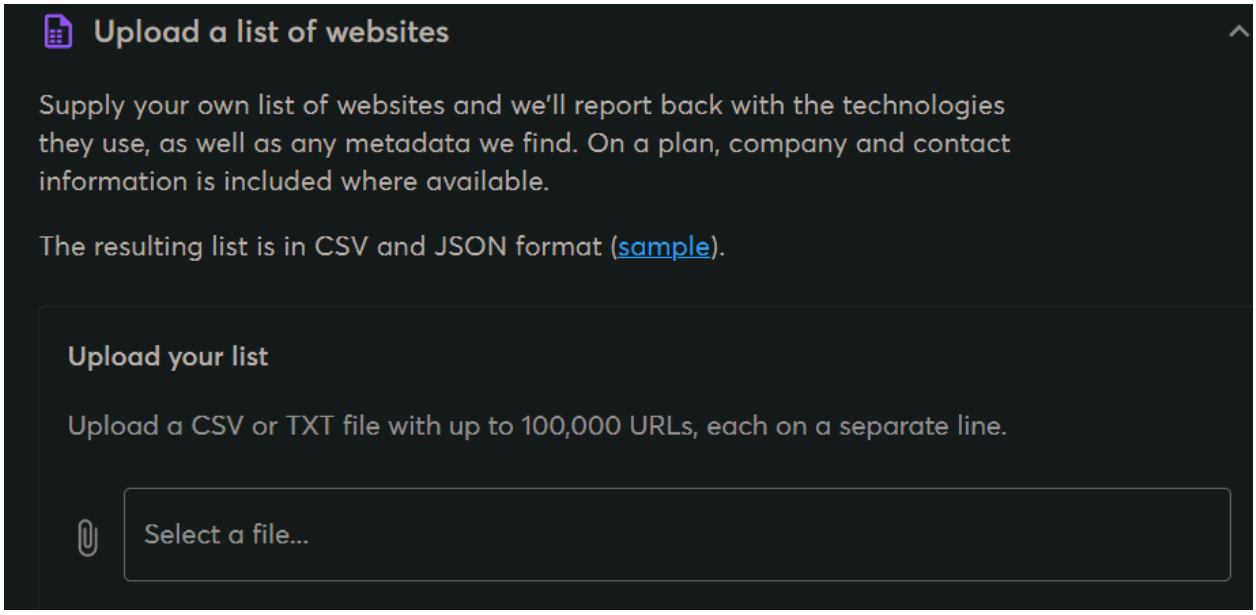
The new developer let out a weak laugh that probably sounded like a sigh. Now that the chronologic expectations involved in the passage of time, all narrative restraint and divisions across literary genres had seemingly been lost, he could gladly without a need for understanding its purpose. His faith had never been stronger.

The penance endured by this developer, with a resolve strong enough not to blast this information out to anyone that would understand, allows you to use the tool today. So beyond some pretty comprehensive fingerprinting abilities, Wappalyzer can also instigate some fairly bizarre tangents when it exposes some dusty, dark secret like the one I just shared with you.



The inception-like screenshot above inspired this short horror masterpiece, thanks for reading. Seeing it is still chilling, even after all these minutes.

As I cautiously dug a little deeper into Wappalyzer to understand its offerings and nuances, a specific sentence seemed to catch my eye almost instantly, though it was not particularly unique in style or color. Maybe I'm alone on this, but does this sound like something you should get started on immediately?



If you looked too quickly, you might have missed the part where it says "100,000 URLs" but I didn't and I got right to work preparing one by combining lots of bounty platforms and parsing various scopes together into a 100K megascope. So I'll spare you the penance and let you know that this sassy piece of code is very sensitive, dropping errors that cannot be ignored if one of your 100,000 subdomains has a comma in it or is prepended with "http". It could just ignore that line and keep it moving, but it forces you to go back into the file, edit it out and resubmit, only to be fed where the next problematic line happens to be. The best part was when I put that whole list together after 25 minutes before it hit me with an error saying "*Free users can only track 50 subdomains*". I had no choice but to do a 360 degree pirouette and moonwalk away from my computer.

Overall, this tool is quite good, maybe the best at what it does, but

power like this doesn't come without a price and I paid for it with a double scoop of horror and anguish.

Another browser option is to use BuiltWith.com again, though I think the command line tools are the most useful for fingerprinting. I suppose if some gnarly CVE is released and you make a mad scramble to BuiltWith to see a list of every company ever using it, then parse the 2% (probably less) that may possibly have an active bounty program. If this sounds like you, there are better ways to live, my friend.

My great adoration for **httpx** reappears, as you can add the **-td** (or **--tech-detect** parameter) and get a nice little blurb about your target's server, framework, CMS - though that is not this tools intended purpose but it is useful when you're feeling lazy or have a lot of info to parse quickly. If you annihilate it with parameters like I always do, you get a good starting point on its tech. Here's a random, recent example below:

https://www.lynxexpression.com [301,200] [] [610554] [text/html] [Lynx: Men's Grooming, Lifestyle…] [] [vhost] [pipeline] [http2] [AS16625, akamai-as, US] [23.7.54.140] [san22.kona.unilever.com.edgekey.net] [Adobe Experience Manager,GraphQL,HSTS,Java] [https://www.lynxexpression.com/home.html]

You can find my primitive and extremely based httpx script that resolves, fingerprints, splits output into clean, pipe-ready files based on status code and usage at the link below:

https://gist.github.com/scumdestroy/c95622e426f52ed497f1483926ffb4bb

An old favorite that still works quite well is **whatweb**, rising to such fame you can even find it in the default Debian apt package

manager (and of course, Kali, Blackarch and Parrot). Syntax is simple and have always used the maximum enumeration aggression setting of "3".

whatweb -a 3 analogworship.com

Whatweb is a forgiving beast, you will make whether you include the full URL or just the domain. You can also use **-iL list-of-guys.txt** to fingerprint a list of domains or whatever you have in there.

On the more bleeding-edge side of fingerprinting developments, you can find the highly intelligent Praetorian-Inc group with their tool **fingerprints**.

go install github.com/praetorian-inc/fingerprintx/cmd/fingerprintx@latest

Very similar to how whatweb is used, fingerprintx follows the syntax popularized many years ago, setting the standard for fingerprinting tools of the future. It seems pretty similar, but I'm sure it's a lot faster and holds a lot more data and makes absolutely sure it's correct before spouting off some false positive that wastes your time.

fingerprintx -t praetorian.com:80

fingerprintx -l input-file.txt

For reasons I don't fully understand, I always gravitate towards this neat little guy called **webtech**. You can install webtech through pip:

```
pip install webtech

webtech --update-db
```

Then we keep our brains smooth and not risk reading a single word in the README file, because its as easy as this:

```
webtech -u werewolf.fi
webtech --ul list-of-questionable-online-vendors.txt
```

If you're into having more control than expected, you can add **`--rua`** for random-user-agent or **`--user-agent "Broken Microwave, Made in Germany 1939"`** to keep the WAFs on their toes. I'm sure most have never faced a nearly century-old microwave, powered by the dreams of men overtaken by occult power fantasies and focused that chaotic energy into such an impressively engineered microwave.

Fuzzing with Wordlists

While an overwhelming majority of hunters, pentesters and recon enthusiasts use wordlists with tools like Dirsearch, FFUF and Burp Suite's Intruder, they are seldom utilizing the breadth of techniques available to them. Using a peasant's common wordlist intended for quick and general endpoints and firing it at a large organization's server is akin to throwing a wet marshmallow at a phalanx of surgically enhanced cyborg mercenaries; ineffective and generally more sad than funny.

First, let's chop it up about fuzzing endpoints with wordlists, the hunter's often first attempt into some potentially sensitive discoveries (and sadly, some hunter's last or only attempts as well). We all run the common "*raft*" lists or maybe "*dicc.txt"* and this is fine for CTF boxes, obscure VDP programs or to get a quick understanding of a few endpoints of the application. For some sweet non-duplicate findings, you should really be doing these few things...

1. **Fuzz each directory you find recursively** (you'll really need a cloud box or it will be too painful and slow to continue on your

path)

2. **Use custom wordlists** created by sourcing things related to your target.
3. **Use targeted wordlists** based on fingerprinting your target's server, framework, technology, target industry (financial, medical, tech). For example, after fingerprinting your target, you'd use an Apache (server) wordlist, a Wordpress (CMS) wordlist, an API wordlist and relevant language wordlist (PHP for Wordpress).

If you don't have any sort of massive, sprawling data dungeon consisting of wordlist files yet, you can fatten your stockpile very quickly by cloning the following repos.

For the hyper-lazy: You can run this script, which will grab quite a lot of popular and much less popular wordlists, including many for cracking passwords and others for a little of everything.

https://github.com/shifty0g/wordlist-tools/

Next is my secret wordlist repo, many of which are completely custom or combine a handful of small, obscure and interesting lists.

https://github.com/scumdestroy/warlusts

Some fat and solid packs below too:

https://github.com/0xPugazh/fuzz4bounty

https://github.com/Karanxa/Bug-Bounty-Wordlists

https://github.com/berzerk0/Probable-Wordlists

https://github.com/trickest/wordlists

And of course, the GOATs:

https://github.com/danielmiessler/SecLists

http://wordlists.assetnote.io/

https://github.com/six2dez/OneListForAll

If you grabbed all the above, you are probably set for life. Just supplant them with your own creations. *Share them only after you have exhausted them on every bounty program in existence though!*

For finding endpoints, it is easy to get lazy or forgetful with how comprehensive your fuzzing can be. The hunters that get the goods take it just a pinch further than the average hunter. Do your best to include fuzzing beyond the standard wordlists and include the following as well**:**

- **Javascript wordlists.** This can either be custom or default wordlists with the ".js" extension added (**"-e .js"** in dirsearch and ffuf). These files often hide in folders like

domain.com/assets/js/FUZZHERE.js or domain.com/static/js/FUZZHERE.js.

- **Documents (xls,xlsx,doc,docx,pdf,...).** Again, default or custom wordlists work best here with the extension flag added. You can then take it further by scanning for more details with `exiftool` (**`"exiftool -a weakdoc.docx"`**). There is also **metagoofil** by opsdisk (https://github.com/opsdisk/metagoodfil) that automates some of this process for you through dorking but you'll need to set up some proxies to choose this route because Google likes to violently force feed anyone acting slightly suspicious with Captchas.

Finally, I'll mention Ademking's "Repolist" (https://github.com/Ademking/repolist), which I imagined some time ago but was too untalented and distracted to create at such a high caliber of quality (or any caliber of quality). Very cleverly, it takes a Github repo URL as an input and creates a wordlist based on the files within it. This is beyond helpful for any targets using open-source components and CMS that you can identify through fingerprinting. To use this silver bullet to slay your werewolves of choice:

```
pip3 install repolist
```

Very simple syntax seen below:

```
repolist -u "https://github.com/octobercms/october" -o
octobercms_wordlist.txt
```

You can even pipe it into **ffuf**, to immediately use it as your wordlist..

repolist -u "https://github.com/WordPress/WordPress" | ffuf -u "http://example.com/FUZZ" -w -

If you want to trim some useless fat, you can stick to a certain branch (**-b BRANCH_HERE**) or just directories (**-d**) or files (**-f**) exclusively.

FUZZING TARGET II : PARAMETERS

Do not be satisfied with simply discovering endpoints, as each can harbor hidden parameters that can be discovered through Javascript or they may be completely hidden in inaccessible code, leaving your only option for exposing them to fuzzing.

There aren't too many hidden gems to guide you towards where hidden parameters reside, as they can potentially hide in any endpoint. Using a bit of deductive reasoning and experience, you'll pick up a sense for looking at the more interactive endpoints to target. There probably isn't much point to fuzzing every single blog entry for new parameters as you'll typically get the same useless results, but endpoints that involve customizable features, ancient code and the usual horrorshow locations that other vulnerabilities tend to reside are best. You always have the option to be a madman and fuzz every endpoint too.

Of course, awesome wordlists for parameters are more effective so just pull the fatties out of my Warlusts repo mentioned earlier and add to it by extracting your discovered URLs through spidering with Katana or

Burp Suite and using unfurl to isolate them. Enough of this.... conceptual talk... let's get to the tools, my friends.

- ImAyrix has a neat tool called **fallparams**, installable like so:

go install github.com/ImAyrix/fallparams@latest

It creates custom wordlists from taking an input of URLs, crawling and analyzing source code, saving you time and killing you with kindness.

fallparams -url "https://nightmare.mit.edu" -crawl -output nightmare-params.txt

- Sh1Yo's **x8** is probably the fastest parameter fuzzer I've found yet, just pick your URL and a wordlist of parameters and it will output any interesting responses it finds (different content-length, status code, reflections) when parameters are added and removed. Since it's written in Rust, it's *fast* but not always fun to compile, so I just download the release like a smoothbrain.

https://github.com/Sh1Yo/x8/releases

Syntax is what you probably expect at this point too.

x8 -u https://kvlt.fi/endpoint.php -w params.txt

The output of this tool is excellent as well, as it shows you how the addition of the newly discovered parameter changes the content-length of the response, the status code and whether the parameter's value is reflected as well.

- S0md3v's **arjun (https://github.com/s0md3v/Arjun)** is probably the most well-known parameter fuzzer used by hunter's today. Using a very clever methodology, the tool attempts about 25,000 different parameters through only around 50 requests over 10 seconds, as well as pulling names for potential parameters from the initial response it receives on initial contact with the target endpoint.

  pip3 install arjun

  The simplest usage can be fired off like so:

  arjun -u https://drakkar666.com/endpoint.php

  You can change method via **-m POST** or **-m JSON**, or input a list of urls:

  arjun -i urls.txt -o arjun-out.json -m POST

Beyond fuzzing parameters, you should also fuzz their values upon finding a valid one via **Burp Suite;s Intruder** or through **ffuf.** Wordlists to use can be content-specific, depending on whether you are trying to find IDORs, include various injection-type vulnerabilities (SQLI, XSS, XPath, etc..), common values (true, 1, 0,0001) or any combination + encoding.

ffuf -c -v -u https://intigriti.me/site.php?secret=FUZZ -w values.txt

Fuzzing Headers

Primarily used for finding bypasses to 401 and 403 status codes, there are some neat tools that make this easier and also include some bypasses that don't involve fuzzing headers.

- https://github.com/gotr00t0day/forbiddenpass
- https://github.com/lobuhi/byp4xx
- Burp Suite's 403 Bypasser Extension

Esoteric Fuzzing Practices of the Inner Circle

Beyond fuzzing paths, parameters and headers, you can even practice your arts where many hunters routinely forget to take a look. |

With Burp Suite's Intruder or using FFUF's **-request** feature, you can also set your target to fuzz the HTTP method (GET, POST and beyond), the HTTP version (1.1, 2.0 and not much more) or the User-Agent value.

Brute-Fuzzing Authorization Headers

A 401 status code is used to prevent unintended users from accessing

an online resource. After a valid user logs in to the site, an "Authorization" header is attached to each request for the remainder of their session, using the syntax seen below:

Authorization: Basic admin:password

If seeing a password in plaintext makes you feel uneasy, don't worry! The credentials are encoded in base64, providing a very frail and brittle coat of protection, which is just one reason it is no longer recommended for protecting modern web assets. If a site is implementing this type of security, it is likely outdated and may have other zany security issues as well. Brute forcing these headers should not cause any accounts to lockout, so you can fire off as many requests as you feel is appropriate when you fuzz directories on the server.
To start trying to crack these, we need to create a list of common usernames and passwords in the format below and save it as a text file.

admin:password
manager:secret
guest:guest
Administrator:changeme

Then you just need to base64 encode each line and use it as your fuzzing wordlist in Intruder or ffuf. Put as many as you feel is reasonable - generally mixing and matching 100 common usernames to 100 common passwords will leave you with 10,000 requests. I'll let you decide if that's OK to fire at your target, no judgments on my end.

No one should have to do this manually and if you don't even want to learn how to automate this process, that is totally fine too.. Probably not the ideal attitude to hold in this field, but again ,no judgements.. Just use this top secret script I made instead..

https://gist.github.com/scumdestroy/219ff280392dbab1923e4f6024d583ee

Gather your users list and your password list and run it like this:

401WordlistTool.py users.txt passwords.txt

It will give you an output file containing base64-encoded blobs of every possible combination of users and passwords. Then just run ffuf like so:

ffuf -H “Authorization: Basic FUZZ” -w output.txt
-u http://target.com/secret -fc 401 -c -v

“Please be responsible with this tool”.
But no judgements, my friend.

Connections Through Analytics Tags, Google Tag Manager and Other Tracking Services

Generally found in Javascript, oftentimes appearing via the syntax "**`google-gtm.js?id=<STRING>".`** Once you find this Google Tag Manager ID, you can head to `https://www.googletagmanager.com/gtm.js?id=<INSERT_ID_HERE>` and find extra endpoints, API keys and more interesting items. You can also run a dork that looks like "**inurl:gtm.js?id=**" to find some nice pages with potentially sensitive info or hidden endpoints and parameters.

You can also try to search for the unique identifier, a public (or private) API key you discovered on Google, Github, Gitlab or the aggregate code harvesting search engines.

One service you can search directly for a Google Analytics tag is SpyOnWeb (https://spyonweb.com/) and find related subdomains or domains owned by the same organization.

Uncovering Cloud-Based Assets

This section will discuss a few ways you can leverage various tools and services to discover hidden assets your target may be utilizing. Sometimes, in bug bounty, these won't be explicitly mentioned in the scope details, however, if you find something that shows a gross violation on user privacy or sensitive information that can be used to damage an organization's reputation, trust or digital assets, they will generally reward you for providing this information to them.

The main providers of cloud assets are Amazon's AWS, Google's GCP and Microsoft's Azure and you can occasionally use dorks to find your target's cloud assets.

site:http://s3.amazonaws.com "target[.]com"

site:http://blob.core.windows.net "target[.]com"

site:http://googleapis.com "target[.]com"

site:http://drive.google.com "target[.]com"

You can also try these dorks without adding the ".com" suffix, just be aware that not everything you find will be related to your target so use your best judgment and an elevated form of common sense before you hype yourself up for disappointment.

A killer service that is essentially a Shodan for cloud assets is Gray Hat Warfare, who provide an impressive and extensive S3 bucket database you can search at https://buckets.grayhatwarfare.com Using the free version of the service shows you a fraction of the results and you'll often be teased with hundreds or thousands of locked bucket assets.

I'm sure your mind is thinking, "I'd like to download every IP address belonging to a cloud provider and scan their entire range for hosted content relevant to my interests". Luckily, that is exactly what I'm about to show you how to accomplish.

The hero behind the site https://kaeferjaeger.gay does a weekly scan of the four biggest cloud providers and pulls all of the SSL certs within this range of IP addresses. You can visit the aforementioned domain and pull these lists for yourself, the size of them is actually much more reasonable than you probably expect.

Once you have those saved to your local workspace, you can run through them and extract whatever domain you are targeting with a one-liner like this:

```
cat *.txt | grep -F ".target.com" | awk -F'-- ' '{print $2}'| tr ' ' '\n' | tr '[' ' ‘| sed 's/ //'| sed 's/\]//' |
```

```
sort -u
```

You can also get a list of IP ranges from Amazon themselves, then use it for your own processing to discover assets.

https://ip-ranges.amazonaws.com/ip-ranges.json

From here, you could pull all of the US based ranges used for hosting with the following command.

```
jq -r '.prefixes[] | select(.service=="EC2") | select(.region=="us-*-*") | .ip_prefix' < ip-ranges.json | sort -u > us-aws-ranges.txt
```

Next, we can feed it to masscan and rip all of the “HTTPS” resources into a neat file.

```
sudo masscan -iL us-aws-ranges.txt -oL us-aws.out -p 443 --rate 200000
```

```
awk '/open/ {print $4}' us-aws.out > aws-443s.txt
```

Finally, we can send it to a tool like tlsx and end up with our own list of domains to query locally at our convenience and discretion.

```
cat aws-443s.txt | tlsx -san -o tlsx-aws-ranges.txt
```

With this list, we can now easily grep for our target organization from it.

```
cat tlsx-aws-ranges.txt | grep target.com
```

There's also various tools that will search S3 buckets for you, based on a keyword. This is very similar to subdomain scanning and can end up with some neat finds. Misconfigurations used to be viciously rampant in this area, due to some very insecure default settings but a few years ago, Amazon realized that many of their customers are smooth brains and catered to them by changing this. Still, some people go out of their way to make their assets insecure and there are still interesting things being discovered here once in a while. There's a lot of tools for fuzzing the internet for S3 buckets, most of which are not in active development since AWS increased their security standards, but I'll note a couple cool ones here.

Redhuntlabs' "Bucketloot" is a few levels above the rest, with its ability to scan through AWS, DigitalOcean, GCP and custom domain bucket resources with a thick list of regexes that will alert you to interesting and sensitive discoveries. You can get it at:

https://github.com/redhuntlabs/BucketLoot

Another excellent and very different option is sa7mon's "S3Scanner", capable of searching through AWS, DigitalOcean, DreamHost, GCP, Linode, Scaleway and custom bucket assets as well. It is actively kept updated andn functional, as well as being used in a few well-known frameworks like ReconFTW, Axiom and ReNgine. It is also written in Go, which is so much better than Python when it comes to needing fast,

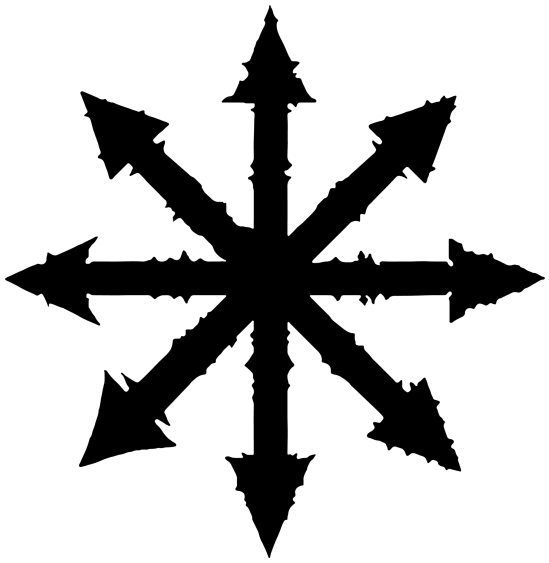
multi-threaded workflows, which fuzzing definitely falls under. Grab it at:

https://github.com/sa7mon/S3Scanner

Usage involves giving a keyword or list of keywords (words related to your target, most likely the name of the organization and domains it operates, sans the TLD).

```
s3scanner -bucket Weyland-Yutani
```

Add the "**-enumerate**" flag if you want the tool to parse all the contents of discovered buckets as well.



Miscellaneous Tips that Transcend Existing Classification Standards

For this final section, I'll drop some juicy recon lore that didn't fit super cleanly into any other section of the book. Thanks for reading this far and I hope your baskets are filled with low hanging fruits for a long and prosperous time.

- **Searching for Copyright Strings:** On the bottom of many websites, exists some copyright in the footer. You can grab that bad boy and slap some quotation marks around him and funnel that info into a Google search. This will help you discover both subdomains and horizontal targets. You can totally swap the year and repeat the search multiple times, to find older and neglected targets to assault.

Google search: **“©2024 Hilton” inurl:Hilton**

- **Sweet Strings Subsisting Submerged:** After spidering, fuzzing and pulling your endpoints from third-party sources, you can be really thorough and save the output of all the valid endpoints into a folder (or just gather them all up in your Burp file). Recon grandmaster Orwagodfather blesses us with the following tip: search for these strings and hope to find some lucky scores. **“Drive.google” , “docs.google”, “/spreadsheets/d/”** and “**/document/d/”**. You could also try to find these as dorks by searching for each one along with “**site:target.com**” too, but there’s a chance that you might miss some that are mentioned in a Javascript file that is hosted on a CDN or something.

- Not exactly sure where to fit this recon tip but I think it will be the underdog golden goose for someone on some day, though I haven’t figured out the specifics yet. If you were attuned to the same vibrational frequencies as me, you’d know this thing will be significant (just not in an overwhelming majority of scenarios). It’s a search engine with a unique filter, called MillionShort (https://millionshort.com/). It removes the first one million commoner normie results from your search, because nobody is browsing or subtractive-dorking their way to the contents found on page 10,000. Maybe use this to find some very common-sounding but actually very rare subdomains of

something like Yahoo? Make it your deus ex machina and tell me about it if my visions have come true yet.

- It's extremely unlikely that you don't already know this, but most domains have a **/robots.txt** endpoint that contains pages the domain owner prefers not to index into search engine results. More exciting, is the **.DS_Store** file you can sometimes find as an artifact left on Apple computers. Then, you can use a tool like https://github.com/lijiejie/ds_store_exp to extract information from the file, which can often disclose paths on the domain used during development.
- Another place to extract additional URLs during recon is from APK files (Android applications). There's various places to get APK files, but I've found APKpure and APKmirror to be trustworthy and comprehensive. A great tool for extracting the URLs is APKURLGrep, available at https://github.com/ndelphit/apkurlgrep
- You can also extract EXIF data from PDF documents and photos hosted on your target using **exiftool**. Mileage from this method will probably be fairly unrewarding, but it could point to a clue that leads to another clue and ends somewhere interesting eventually.

RECON BOSSMAN CHECKLIST

Note: Some steps may not apply to your target.

Check Acquisitions

- Company blogs, Websites and Social Media Pages
- Financial Reports and SEC Filing Data
- Newsletters and Private Reports
- Misc Sources (Press releases, job postings, etc..)
- Third Party Sites (intellizence.com, owler.com, crunchbase.com, mergerlinks.com, mergr.com, aleph.occrp.org)

ASN (if applicable)

- bgp.he.net
- discover if target actually owns an ASN and hosts their own domains
- amass intel -org ORGANIZATION_NAME
- amass intel -asn ASN_NUMBER
- nitefood's tool "asn", projectdiscovery's "asnmap" or harleo's "asnip"
- Shodan dork asn:asXXXXXX

- Censys dork autonomous_system.asn:XXXXXX

or autonomous_system.name:"ORGANIZATION"

- zoomeye asn:XXXXXX

- Fofa: asn="XXXX"

Reverse Whois

- whoxy.com/reverse-whois

- gwen001's "related-domains"

CIDRs and IPs

- asnmap -d domain.com

- nmap --script targets-asn --script-args targets-asn.asn=XXXXXX

- prips 10.10.10.0/24 | hakrevdns

- prips 10.10.10.0/24 | hakip2host

- prips 10.10.10.0/24 | hakoriginfinder -h URL

- ImAyrix's "cut-cdn"

Ports

- masscan

- nmap

- naabu
- rustscan

Shodan

- ssl.cert.subject.cn:"domain.com"
- ssl:"Organization Inc."
- http.html:"ua-XXXXX" (Google Analytics Code)
- Favicon Hash tools (MurMurHash)
- http.favicon.hash:XXXXXXXX
- nrich

Censys

Nelas.io

Fullhunt.io

- http_favicon_hash:XXXXXXX
- domain:domain.com

Onyphe.io

Viz.Greynoise.io

Analyzeid.com

Dnslytics.com

Google Dorks

- site:domain.com -site:www.domain.com

- Google Groups, Docs

- Dorks looking for config files, backup files, api documentation

- Tools like go-dork or dorksearch.com

Github Dorks

- Tools like gwen001's github-subdomains, github-endpoints, github-regex, michenriksen's gitrob or trufflesecurity's trufflehod

- Manual searching

Subdomains

Passive Discovery

- Shodan, Censys, Third-Party Services

- Query rapiddns.io, JLDC, riddler.io, Archive.org, DNSDumpster and more

- Passive subdomain tools like projectdiscovery's "subfinder", OWASPs "amass", Findomain, hueristiq's "xsubfinder", ARPSyndicate's "puncia"

Certificates

- crt.sh

- Censys Certificates Search (443.https.tls.certificate.parsed.subject.common_name:domain.com)

- cero

- CSP or 0xbharath's domains-from-csp tool

- SPF or 0xbharath's assets-from-spf tool

- ui.ctsearch.entrust.com/ui/ctsearchui

ProjectDiscovery's "Uncover"

Active Subdomain Hunting

- Zone Transfers

- Brute Force

- Permutations

- VHOSTs

Content Discovery

- Passive URL collection (archive.org, urlscan.io, commoncrawl.org, alienvault.org)

- xnl-h4ck3r's "waymore", devanshbatham's "paramspider", hueristiq's "xurlfind3r" or lc's "gau"

- Spidering website

- projectdiscovery's "katana", jaeles-project's "gospider", Jason Haddix's Burp Suite Passive Collection while you browse target method

- Extract endpoints from JS files

- Pull from collected URLs and feed to 003Random's "GetJS"

- Grep for interesting strings or use tomnomnom's "gf"

- use 0xsha's "GoLinkFinder", kacakb's "jsfinder", leddcode's "Aranea", channyein1337's "jsleak" or BishopFox's "jsluice"

- Pull source maps

- Hunt for secrets

- m4ll0k's "secretfinder", edoardott's "cariddid"

Fingerprinting

- projectdiscovery's "httpx", urbanadventurer's "whatweb", praetorian-inc's "fingerprintx" or rverton's "webanalyze"

- use Wappalyzer, builtwith.com or retire.js

- Fuzzing endpoints with wordlists

- General wordlists

- Targeted wordlists based on fingerprinting data

    - Custom wordlists based on content

- **Fuzzing Parameters**
    - s0md3v's "arjun" and Sh1Yo's "x8"

- **Fuzz Headers to Bypass 401s and 403s**

- **Google Analytics Tags**

- **Cloud-Based Asset Enumeration**

- **Misc Tips**
    - Copyright and Footer search in Google
    - Extract endpoints from APK files
    - EXIF data extraction from PDFs and Images

RESOURCES

A ***special thanks*** to the following human beings that have selflessly contributed to the tools, techniques, research and concepts that shape and progress this field. Their penchant for creation and destruction across the cyber realm ensures our safety, while simultaneously establishing the reality that no one of us will ever be safe. I highly recommend you follow their research and their lives for knowledge and inspiration across Github, Twitter, their presentations, blogs and anywhere else you find them and their work. Again, thank you - without y'all this book would be a pamphlet about the best ways to guess passwords and subdomains.

Hahwul https://www.hahwul.com/
"Hahwul" at Github, Twitter and Hackerone.

Harsh Bothra https://harshbothra.tech
Harshbothra_ on Twitter
Harsh-bothra on Github
https://hbothra22.medium.com/

Project Discovery https://projectdiscovery.io/
Projectdiscovery at Github

Jason Haddix
Jhaddix at Github, Twitter
https://www.youtube.com/c/jhaddix

Zseano https://bugbountyhunter.com
Zseano at Github, Twitter, Hackerone
https://www.youtube.com/c/zseano
https://zseano.medium.com/

hakluke https://hakluke.com/
Hakluke on Github, Twitter, Youtube and Medium

Harleo https://harleo.me/
Harleo at Github
_harleo at Twitter

Gwen001 https://offsec.tools/
https://10degres.net

Gwen001 at Github
Gwendallecoguic at Twitter

Trickest https://trickest.com
Trickest on Github
Trick3st on Twitter

Six2dez https://pentestbook.six2dez.com/
Six2dez on Github
Six2dez1 on Twitter
Six2dez_ on Hackerone

Dwisiswant0
Dwisiswant0 on Github, Twitter, Medium and Hackerone

Nahamsec https://nahamsec.com
Nahamsec at Github, Twitch and Twitter
https://www.youtube.com/@NahamSec

Tomnomnom https://tomnomnom.com/
Tomnomnom on Github and Twitter

Owasp-amass https://owasp.org/www-project-amass/
Owasp-amass at Github

Nitefood
Nitefood at Github

Imusabkhan
Imusabkhan at Github
Musab1995 on Twitter
https://www.youtube.com/musabkhan
https://medium.com/@imusabkhan

M4ll0k
M4ll0k on Github, Twitter, Bugcrowd and HackerOne.

KathanP19
KathanP19 on Github, Twitter and Medium

Xnl-h4ck3r
Xnl-h4ck3r on Github, Twitter and Youtube

Devanshbatham
Devanshbatham on Github
0xAsm0d3us on Twitter

ImAyrix
ImAyrix on Github and Twitter

Robertdavidgraham
Robertdavidgraham on Github

Vulnerscom https://vulners.com/
Vulnerscom on Github and Twitter

Viralmaniar
Viralmaniar on Github
ManiarViral on Twitter

Jakejarvis https://jarv.is/
Jakejarvis on Github and Twitter

Sushiwushi
Sushiwushi on Github
Sushiwushi2 on Twitter
https://sushiwushi.notion.site/Richie-s-Audit-Portfolio-ecac81e9bfcd46fc94ca8715d234f7d6

Trufflesecurity https://trufflesecurity.com
Trufflesecurity on Github
Trufflesec on Twitter

Michenriksen https://michenriksen.com/
Michenriksen on Github and Gitlab

Techgaun http://www.techgaun.com/
Techgaun on Github and Twitter

Bishopfox https://bishopfox.com
Bishopfox on Github and Twitter

Tillson https://tillsongalloway.com/
Tillson on Github

Kootenpv
Kootenpv on GIthub

Incogbyte https://rodnt.github.io/
Incogbyte on Github and Twitter

Pirxthepilot https://pirx.io/
Pirxthepilot on GIthub

Edoardottt https://edoardoottavianelli.it/
Edoardottt on Github

Glebarez https://t.me/glebarez
Glebarez on Github
Blegmore on Twitter

0xbharath https://www.disruptivelabs.in/
0xbharath on Github and Twitter

Hueristiq http://hueristiq.com/
Hueristiq0x00 on Twitter

ARPSyndicate https://www.arpsyndicate.io
ArpSyndicate on Github

D3mondev
D3mondev on Github and Twitter

Wdahlenburg https://wya.pl/
Wdahlenburg on Github
Wdahlenb on Twitter

N0kovo https://infosec.exchange/@n0kovo
N0kovo on Github and Twitter

Jobertabma https://hackerone.com
Jobertabma on Github and Twitter

Dariusztytko
Dariusztytko on Github and Twitter

Josue87
Josue87 on Github
JosueEncinar on Twitter

Vortexau http://www.vortex.id.au
Vortexau on Github and Twitter

Bp0lr https://www.micropay.com.ar/
Bp0lr on Github and Twitter

Jaeles-project https://jaeles-project.github.io/

0xDexter0us https://blog.dexter0us.com/
0xDexter0us on Github and Twitter

003random https://003random.com/
003random on GIthub
Rub003 on Twitter

Leddcode https://leddcode.github.io/

Kacakb https://github.com/kacakb

Gareth Hayes https://garethheyes.co.uk/
Garethhayes at Twitter

0xsha https://www.0xsha.io/
0xsha on Github and Twitter

Sirbugs
Sirbugs on Github
SirBagoza on Twitter

Byt3hx
Byt3hx on Github

Robre https://r0b.re/
Robre on Github
R0bre on Twitter

Streaak
Streaak on Github, Twitter, Bugcrowd and HackerOne

Shifty0g
Shifty0g on Github, Twitter and Bugcrowd

Karanxa
Karanxa on Github
ItsKaranxa on Twitter

Berzerk0
Berzerk0 on Github and Twitter

Danielmiessler https://danielmiessler.com/
Danielmiessler on Github and Twitter

Ademking https://ademkouki.site/
Ademking on Github
Kouki__adem on Twitter

Sh1Yo
Sh1yo on Github and Hackerone
Sh1yo_ on Twitter

S0md3v
S0md3v on Github, Twitter and Medium

Opsdisk https://opsdisk.com/
Opsdisk on Github and Twitter

0xPugal
0xPugal at Github and Twitter

Praetorian-inc https://www.praetorian.com/
Praetorian-inc on Github

Ndelphit
Ndelphit on Github
Delphit33 on Twitter
Whoareme on Hackerone

Lobuhi
Lobuhi on GIthub
Lobuhisec on Twtiter

SA7MON https://danthesalmon.com/
Sa7mon on Github

Redhuntlabs https://redhuntlabs.com/
Redhuntlabs on Github and Twitter

Gotr00t0day
Gotr00t0day on GIthub

Using all of this information with efficiency and creativity will make you a recon professional with a bright future. Just remember to be patient, be thorough, check assets over time with an automated method and find some unique methodology that is effective for you.

Jann Moon is a penetration tester and bounty hunter with a deep interest in recon and web hacking. He has worked as an audio engineer, touring black metal musician and an illustrator for clients like Dungeons & Dragons, TMNT, Brutal Truth, Bastard Noise, Noisear and Vermin Womb. He has also worked as a technical writer for Linode, been a cybersecurity expert in court cases and discovered CVE zero-days for fun and profit.

If you'd like to send any questions, ideas, fix errors, pool resources for glorious purpose, collaborate or otherwise, you can reach me at:

jann@scumdestroy.com

https://github.com/scumdestroy

https://scumdestroy.com

Thanks for reading!