How do you approach a target?

samhouston Senior Community Manager — Apr '15

Jumping off of **our interview we did with Fredrik**, I thought it would be great for a discussion about how researchers approach a target. I've heard many different methods, whether it's a certain set of tools and vulnerabilities that people look for when they start, or perhaps something totally different.

Question:

When you're accessing a new target (new to you), how do you approach that target and optimize for success? What tools, processes, research, etc do you do when getting started?

I'd love to hear some thoughts. Shoutout to **@anshuman_bh** for the topic idea.

What is the first step you take to start attacking a website?

Researcher Resources - How to become a Bug Bounty Hunter

How do you avoid duplicates in a Bug Bounty program?

Apr 2015

1 / 11

Apr 2015

17d ago

anshuman_bh — Apr '15

Since I brought this up, I'll start this thread 😄

I started hunting for bugs about an year or so ago. Just like how most bug bounty hunters get started, I too mostly reported low hanging fruits and petty issues that companies didn't really care about in the beginning. Fast forward to 2015, I like to believe that I have improved over the past year or so and I will explain how.

Some things that I have done/observed that has significantly helped me reduce duplicates and get bugs accepted in the first go. I understand this might sound repetitive because it is not rocket science so just bear with me:

1. Start early. As soon as a program is launched, start hunting immediately, if you can.
2. Once you start hunting, take a particular functionality/workflow in the application and start digging deep into it. I have stopped caring about low hanging fruits or surface bugs. There is no point focussing your efforts on those.
   So, let's say an application has a functionality that allows users to send emails to other users.
3. Observe this workflow/requests via a proxy tool such as Burp. Burp is pretty much the only tool I use for web app pentesting.
4. Create multiple accounts because you would want to test the emails being sent from one user to another. If you haven't been provided multiple accounts, ask for it. Till date, I have not been refused a second account whenever I have asked for it.
5. Now, if you are slightly experienced, after a few minutes of tinkering with this workflow, you will get a feeling whether it might have something interesting going on or not. This point is difficult to explain. It will come with practice.

6. If the above is true, start fuzzing, breaking the application workflow, inserting random IDs, values, etc. wherever possible. 80% of the time, you will end up noticing weird behavior.
7. The weird behavior doesn't necessarily mean you have found a bug that is worth reporting. It probably means you have a good chance so you should keep digging into it more.
8. There is some research that might be required as well. Let's say you found that a particular version of an email server is being used that is outdated. Look on the internet for known vulnerabilities against it. You might encounter a known CVE with a known exploit. Try that exploit and see what happens (provided you are operating under the terms and conditions of the bug bounty).
9. There might be special tools that are required. Explore into that, if possible. Remember, Burp is a swiss army knife but you might have to use certain specific tools in certain cases. Always, be aware of that.
10. After spending a few hours on this, if you think you have exhausted all your options and are not getting anything meaningful out of it, stop and move on. Getting hung up on something is the biggest motivation killer but that doesn't mean you are giving up. Get back to it later if something else comes up. Make a note of it.

So, that's it. Above is an example of how I would approach a program. Feel free to ask questions/agree/disagree.

Cheers!

---

maK0 **Apr '15**

Some good advice there for beginners alright^

Something that has worked for me is bounds checking on parameters, pick a parameter that has an obvious effect on the flow of the application.
For example, if a field takes a number (lets call it ID for lulz).
What happens if:
-you put in a minus number?
-you increment or decrement the number?
-you put in a really large number?
-you put in a string or symbol characters?
-you try traverse a directory with .../
-you put in XSS vectors?
-you put in SQLI vectors?
-you put in non-ascii characters?
-you mess with the variable type such as casting a string to an array
-you use null characters or no value

I would then see if I can draw any conclusions from the outcomes of these tests,
-see if I can understand what is happening based on an error
-is anything broken or exposed
-can this action affect other things in the app.

If I find anything the next step is a simple proof of concept of how it can be abused.

Hope this helps 😄

---

geekspeed **Apr '15**

Everything above is all good stuff…I tend to focus on areas where developers have relied on pieces of code written by other individuals and just trusted that it ‘works’ and ‘is safe’. SAML/Oauth integrations are a good target, 3rd party libs…
Other fun places to start: unvalidated redirects, artifacts left behind from sloppy devops practices / integration tools, common framework fingerprints

---

Nahamsec Apr '15

I don’t think @anshuman_bh could’ve said it any better. I’d like to add that also if subdomains are in scope, start with those, rather than going straight for the main site (especially if you are late to the party). So you are looking elsewhere while everyone is pentesting the main target such as a dashboard and so on.

---

planetzuda Apr '15

That’s where we always start at as well. You start at the worst area and work your way up.

---

geekspeed May '15

I guess therein lies the issue – if we all are attacking subs first…who’s getting the main site?

jhaddix-bcebp May '15

A great thread. Seems like the heavy hitters have already discussed some great points that I use as well. Here are some of my notes:

Approaching a Target at a High Level:

1. As mentioned before *.acme.com scope is your friend. Subdomains are notorious for not having the same amount of security focus as the primary site. Subdomain enumeration has been discussed elsewhere in the the forum so i won't do into that here.

2. Don't forget to portscan for obscure services on all hosts. Get comfortable with nmap. Many high severity issues have been found on non-standard ports either via service exploitation or finding even more hosted web servers. A great and hilarious example is **Shahmeer's IIS.net bug** . Also, **don't forget** that there are a multitude of DNS and mail server bugs that might be in scope.

3. Not pertaining to Bugcrowd (I guess it could though) is looking for acquisitions the company has had recently. Brands like Google and Facebook acquire other companies rapidly. These sites usually have a "grace" period for which you cannot test them or are not covered in the scope of the bounty program... **at first**. Keep a calendar and watchful eye on them, as they have undergone a whirlwind of IT Sec auditing from the Google/Facbook/etc team but usually still hold a higher percentage of bugs than normal. A good place to watch would be something like these lists of **Google** and **Facebook's** acquisitions on wikipedia.

4. Focus on site functionality that has been redesigned or changed since a previous version of the target. Sometimes, having seen/used a bounty product before, you will notice right away any new functionality. Other times you will read the bounty brief a few times and realize that they are giving you a map. Developers often point out the areas they think they are weak in. They/us want you to succeed. A visual example would be new search functionality, role based access, etc. A bounty brief example would be reading a brief and noticing a lot of pointed references to the API or a particular page/function in the site.

5. If the scope allows (and you have the skillset) test the crap out of the mobile apps. While client side bugs continue to grow less severe, the API's/web-endpoints the mobile apps talk to often touch parts of the application you wouldn't have seen in a regular workflow. This is not to say client side bugs are not reportable, they just become low severity issues as the mobile OS's raise the bar security-wise.

6. Lastly don't forget to do **OSINT** on your target. I have found many a quirky "bug" doing this.

Examples:

- Comments or paths in a binary/site leading to a developer Github account with creds and private certs.
- Parsed valid usernames of super users from metadata for brute force attacks.
- Discovered vulns already circulating the web that the client was unaware of. On this one there are a number of sources to look at, but there are some pretty public ones that you will want to check first

like; **Xssed.com** , **Reddit XSS - /r/xss** , **Punkspider** , **xss.cx** , **xssposed.org** , twitter, and a plethora of forums that Google may or may not have indexed.

Approaching a Target at an Application Level:

Fingerprinting and mapping are paramount when looking for bugs that aren't shallow. For fingerprinting you need to understand (or at least identify) any frameworks you are testing against. Some quick Chrome extensions and tools here can help with that:

- **Wapplyzer**
- **Builtwith**
- **Retire.js**
- **Ghostery**

These are just some of the methods you can use. There are nmap scripts that are designed for this as well but i find these to be mostly sufficient.

When fingerprinting identifies some sort of COTS software/framework/++ you need to go searching for that versions update list and possibly any **CVE's** or **disclosures** around it. Identifying a CMS is great because there are a few nifty tools to do this whole process that i like:

- **CMSmap**
- **WPScan**

Mapping is the art of finding application paths. In large applications this art becomes a necessity. Traditional knowledge will tell you that your spider or scanner will give you a perfect site-tree to inspect but seasoned testers know that this is simply not true. A full browse of the site while connected to an interception proxy is mandatory. Are there ways to speed this up or ensure completeness? No, not 100% but i do like utilizing something like **Linkclump** to drive exploration.

Besides "walking" the app you need to discover **unknown content**. This is directory brute forcing. Many people will utilize **Dirbuster** or **Burp Suite's Discover Content** here. The problem with that is both those have drawbacks:1) buggy code and 2) poor directory lists. I prefer using **wfuzz** or **patador** with the vastly superior lists from the **RAFT project** (included in the **fuzzdb** and **seclists** project), **SVNDigger** , and **GitDigger** projects. Why are these better? Well, the traditional lists were created by spidering the net and compiling common application paths. The RAFT lists were created by spidering the net's robots.txt files, giving you awesome quick hits on stuff site admins do not want you seeing or testing. With the Digger projects you can find in depth functions becuase thay were created from reversing source code pathing.

So after you have a thorough "feeling" for the site you need to mentally or physically keep a record of workflows in the application. You need to start asking yourself questions like these:

- Does the page functionality display something to the users? (XSS, Content Spoofing, etc)
- Does the page look like it might need to call on stored data? (Injections of all type, Indirect object references, client side storage)
- Does it (or can it) interact with the server file system? (Fileupload vulns, LFI, etc)
- Is it a function worthy of securing? (CSRF, Mixed-mode)
- Is this function a privileged one? (logic flaws, IDORs, priv escalations)
- ++

There are other things I look at like in-depth analysis of session management and authorization but those bugs are usually found pretty quick.

If you've approached a target like above and you truly understand it you will find bugs most of the time. Your troubles will be the next tier of testing which is understanding errors (always pay special attention to what triggered your errors! **New people often forget to analyze these**), getting injections to work, creating PoCs, fixing broken exploit code/scripts, etc.

Anyways, hope that helps! Happy Hacking!

-jhaddix