EC521 Cybersecurity | SubGuardian

Team 1: Wei Chen, Nurassyl Medeu, Jiawei Sun, Jeet Vijaywargi

Instructor: Manuel Egele

GitHub: https://github.com/nmedeu/SubDomainTakeover

1. Introduction

1.1 Project Description

The final project, "SubGuardian," is a subdomain takeover vulnerability detection and prevention tool. First, let us define subdomain takeover. Subdomain takeovers happen due to misconfigured DNS entries, mainly when DNS records for a subdomain continue to point to external services not in use. This allows attackers to register or claim these abandoned services, thereby gaining control over the subdomain and potentially using it for malicious purposes. For instance, assume a domain *example.com* with a CNAME record *blog.example.com* pointing to GitHub pages. Suppose the CNAME points to the deleted GitHub pages, and the DNS record still exists. In that case, the DNS record becomes dangling, and the subdomain becomes vulnerable to being taken over since any adversary can create GitHub pages that the CNAME is pointing to and start hosting their content.

Our tool, *Subguardian*, aims to detect such subdomain takeover vulnerabilities and make preventions based on detection results, as illustrated in the following report. The results of detection and prevention will be recorded in log files for future reference.

1.2 Project Inspiration

This project was inspired by several aspects. First, subdomain takeovers raise security professionals' awareness of the importance of proper DNS configuration. Fully understanding the risks associated with subdomain takeover can help to adopt frameworks to perform regular audits and develop more strict domain management policies. The neglect of misconfigured DNS records

would lead to significant security consequences. By exploiting these vulnerabilities, attackers can launch phishing attacks, distribute malware, and potentially cause data breaches.

Moreover, malicious attackers can exploit these weaknesses by hijacking the subdomain and manipulating it for various nefarious purposes. In a Ferrari subdomain takeover case in 2022, some hackers hijacked an orphan subdomain and hosted their own NFT for sale.

1.3 Importance & Interests

Subdomain takeovers are important to learn about as they are a vulnerability to which you can never be alerted. If a company's subdomain were to be taken over, it is very likely that they would not even know about it until it was too late and had negatively affected their systems or company.

The consequences of a successful subdomain takeover can be far-reaching. Attackers can leverage the compromised subdomain to redirect users to phishing websites that steal login credentials or distribute malware. They can also mask their activities under the legitimate organization's domain, making phishing attempts even more deceptive. Furthermore, if the vulnerable subdomain hosts internal applications or any other services, a successful takeover can grant attackers access to sensitive data repositories, potentially leading to data breaches. This issue extends beyond data loss, as it can also expose organizations to legal and compliance risks. Non-compliance with data protection regulations such as GDPR and HIPAA can result in hefty fines and legal actions.

2. High-Level Overview

2.1 System Structure

The overall tool works in simple steps. It first begins to enumerate user-provided domains using dnsrecon + sublist3r. This enables us to find subdomains and parse through them for our inputs to functions that perform checking for vulnerabilities. This step sets us up for sub-domain vulnerability checking, and it is done in the background, where the user is not able to see how it is enumerating.

After our tool finishes enumeration, we begin to perform sub-domain checking. This involves checking the A-record, MX-record, CNAME-record, TXT-record, and NS-record for the specified

subdomain to detect whether or not it is vulnerable. The implementation details section of this report will provide more information on how specific checks are performed.

We wanted our tool to also act as a prevention tool; therefore, we decided that not only will we let the user know what the vulnerable domains are, but we will also try to actively remove them. One of the ways we could achieve this was by removing the DNS records from the nameserver, if possible. In order to do that, we need to know what nameservers are being used and the API keys that allow us to manage DNS entries. Our tool currently only supports the active deletion of DNS records from Cloudflare. Therefore, if the user is using Cloudflare and our tool provides them with API keys to manage their DNS records, then the tool can remove the DNS records that point to vulnerable end points. This made the "prevention" section of our tool be successful.

Lastly, we wanted our tool to let the user know if it found or did anything. Therefore, our tool creates a log file at the end of all of its processes to let the user know what vulnerable subdomains it has found and which ones it was able to delete.

2.2 Input & Output Description

Subguardian tool is a CLI application that requires the user to provide the argument *-d*, followed by the domain they want to scan. Our tool also supports tailored enumeration, based on the functionality of *dnsrecon* and *sublist3r*. For example, by providing *-y* or *-b*, the tool will not use Yandex or Bing during the enumeration process. Here is an example usage of the tool: "python3 subguardian.py -d bucrib.com.". However, that will not enable the tool's prevention functionality. To do so, the user must pass in *-p* followed by the DNS host they use, such as Cloudflare. In that case, the user will have to provide the tool with the API keys in the .env file, where the format is already specified. Other functionality could be checked by running the program with *-h*.

3. Implementation Details

3.1 Enumeration

For the first step of subdomains and more general DNS record enumeration, we have combined *sublist3r* and *dnsrecon* tools to search potentially hidden subdomains thoroughly. *Subllist3r* is a tool that searches in the open source for subdomains of the target domain and returns a list of those. *Dnsrecon* is a more powerful tool used for DNS reconnaissance, and we utilize its ability to perform zone walks and search for DNS records. We had to filter out the functionality of the two tools so that they only provided us with the information required to assess the website's subdomain takeover vulnerability.

3.2 CNAME Records

Within the domain name system (DNS), CNAME records play a crucial role in managing subdomains. They function by creating aliases, essentially pointing a subdomain to another domain entirely. While this functionality simplifies service integration, it also introduces a potential security vulnerability. Misconfigured CNAME records, or those linked to inactive services, can be exploited by malicious actors for unauthorized domain takeovers. These takeovers can have devastating consequences, compromising domain integrity and exposing sensitive data or systems.

Subguardian tackles CNAME record vulnerabilities through a multi-faceted detection approach. The approach goes beyond verifying record validity as it also delves into the content served at the redirected location:

Service Fingerprinting: *Subguardian* maintains a database called, "can-i-takeover-XYZ," which can be found here, of known service fingerprints and unique identifiers for various web services. The database is a GitHub repository, which allows everyone to provide it with potentially vulnerable third-party services, which it then displays on the readme file as a table. By comparing the content served under a CNAME record against these fingerprints, the *Subguardian* can assess if the intended service is active and functioning correctly. Deviations from expected fingerprints could indicate a misconfiguration or potential takeover attempt.

Error Pattern Recognition: The tool actively scans web content for specific error messages that are indicative of misconfigurations on the redirected server. These error messages can provide early warnings of potential security weaknesses before attackers can exploit them.

HTTP Status Code Analysis: *Subguardian* analyzes the HTTP status codes returned by the redirected server. Status codes like 404 (Not Found), 410 (Gone), or 403 (Forbidden) can signify potential issues, such as a non-existent resource or unauthorized access. By analyzing these codes, *Subguardian* can further gauge the likelihood of misconfiguration or a takeover opportunity, prompting investigation and remediation.

To check the CNAME records, we have a cname_check.py file within our project that is called to perform the necessary checks from our cli.py once the enumeration process is complete. It first loads service fingerprints and error patterns from JSON and text files. The cname_check function takes input from a list of subdomains and iterates over each subdomain. It calls each subdomain the check_http_response_and_content function, passing the subdomain along with the loaded service fingerprints and error patterns.

Within the check_http_response_and_content function, it attempts to make an HTTP request to the subdomain and fetches the page content. Then, it tries to resolve the CNAME record for the subdomain. If a CNAME record is found, it is compared against the loaded service fingerprints. If a match is found between the CNAME record and a fingerprint, it searches the page content for a corresponding fingerprint match. If both conditions are met, indicating a potential vulnerability, it constructs a dictionary entry with the subdomain as the key and a list of vulnerability reasons as the value.

3.3 TXT Records

Similar to CNAME record checking, we have a file called *txt_check.py* that is also called from our cli.py after enumeration is completed. It first loads vulnerability patterns for TXT records from a JSON file using the load_txt_patterns function. This function attempts to open and read the JSON file

containing the TXT patterns, handling possible exceptions like file not found or JSON decoding errors.

The txt_check function takes a list of TXT records as input and iterates over each record. For each record, it compares its content against the loaded TXT patterns for known vulnerability indicators. If a match is found between the TXT record content and a pattern, it constructs a vulnerability reason string indicating the matched pattern and possible vulnerability.

The function accumulates vulnerability reasons for each subdomain in a dictionary called vulnerabilities, where the keys are the subdomain names and the values are lists of vulnerability reasons. Finally, it returns this dictionary containing information about potential vulnerabilities found in the TXT records of the subdomains.

3.4 MX Records

Similar to CNAME and TXT records, we have a mx_check.py file that checks the MX records.

The check focuses on SMTP connection status and domain expiration status. It checks each MX record in the provided list and performs the following checks:

SMTP Connection Check: It attempts to establish an SMTP connection to each MX record's exchange server on multiple SMTP ports (25, 587, and 465) using the `check_smtp_connection` function. If the connection is successful, it returns a message indicating the successful connection. If not, it collects error messages indicating the failure reason.

Domain Expiration Check: It checks if the domain associated with each MX record is expired using the `check_if_expired` function. If the domain is expired, it adds a corresponding reason to the vulnerability list.

The `mx_check` function aggregates the vulnerability reasons for each MX record and constructs a dictionary where the keys are the exchange servers of the MX records, and the values are lists of vulnerability reasons. Finally, it returns this dictionary containing information about potential vulnerabilities found in the MX records.

3.5 NS Records

We have *ns_check.py* to check for NS records. The code is responsible for conducting NS (Name Server) record checks, focusing on various aspects such as orphaned records, domain expiration status, and the operational status of name servers. There are helper functions that help conduct checks on the overall NS record.

Orphaned DNS Record Check: The `check_if_orphaned` function examines whether the provided NS record is orphaned, meaning it does not resolve to any IP address. It checks for both IPv4 and IPv6 address types and returns a boolean indicating whether the record is orphaned and the IP address type.

Domain Expiration Check: The `check_if_expired` function checks if the domain associated with the NS record is expired by querying WHOIS information. It returns a boolean indicating whether the domain is expired.

NS Server Status Check: The `check_ns_status` function verifies the operational status of the name server by attempting to resolve a domain using the specified name server. It returns a boolean indicating whether the name server is down, along with the IP address and its type.

The `ns_check` function iterates through the provided NS records, performing these checks for each record. If any vulnerabilities are detected, such as orphaned records, expired domains, or non-responsive name servers, it constructs a dictionary where the keys are the target name servers, and the values are lists of vulnerability reasons. Finally, it returns this dictionary containing information about potential vulnerabilities found in the NS records.

3.6 A Records

This code for conducting A record checks, focuses on various aspects such as web server availability, common port scanning, and WHOIS information retrieval. The helper function are set up as follows:

1. WHOIS Check: The `check_whois` function attempts to retrieve WHOIS information for a given IP address. If WHOIS information is successfully obtained, it returns `True`; otherwise, it returns `False`.

2. Web Server Availability Check: The `check_web_server` function checks if a web server responds at the specified IP address by attempting to make an HTTP/HTTPS request. If a response is received, it returns `True`; otherwise, it returns `False`.
3. Common Port Scanning: The `scan_common_ports` function scans common ports on the specified IP address to check for open ports. It reads port numbers from a file and attempts to connect to each port. If a connection is successful, the port is considered open, and its number is added to the list of open ports.
4. A Record Check: The `a_check` function iterates through a list of A records. For each record, it first checks the availability of the associated web server. If the web server is not available, it performs additional checks, including common port scanning and WHOIS information retrieval. If any vulnerabilities are detected, such as no open common ports or WHOIS check failure, it constructs a dictionary where the keys are the A record names, and the values are lists of vulnerability reasons. Finally, it returns this dictionary containing information about potential vulnerabilities found in the A records.

3.7 Prevention

The prevention capability by deleting vulnerable sub-domains was one of the goals of the overall functionality of our tool. First, if the user passes the argument *-d* followed by the used DNS host, the tool will check if the API keys were provided in the .env file. If not, then an error will be raised. Otherwise, it will proceed with the scan. The tool proceeds to iterate through the vulnerable subdomains, returned by the DNS record checks, and delete them by issuing appropriate API requests. This automated process streamlines security management efforts, ensuring swift and efficient remediation of identified vulnerabilities.

4. Evaluation

4.1 Evaluation Target Selection and Results

To evaluate our tool, we have tried replicating the most common DNS record misconfigurations with our domain. Most testing was performed with CNAME records, as they are used to point to third-party services that lead to takeovers. Therefore, we created CNAME record subdomains that point to non-existent domains and third parties, such as Shopify, GitHub pages, SmugMug, AWS, and several other services. Following our initial setup, we conducted rigorous testing to assess the tool's efficacy in detecting and mitigating simulated vulnerabilities. Utilizing the CNAME check Python script, we systematically performed comprehensive scans across our domain infrastructure. Each subdomain underwent the checks described to identify any indications of compromise. By juxtaposing the tool's findings against deliberately crafted misconfigurations, we gained valuable insights into its effectiveness in identifying and remediating potential security weaknesses. These tests were instrumental in evaluating the tool's resilience and dependability in bolstering domain security against prevalent DNS misconfigurations. We found a high success rate at detecting if a website has the potential to be taken over.

We have also conducted specific tests on DMARC policies within TXT records. DMARC is a crucial email authentication protocol that helps domain owners protect their domains from unauthorized use, such as email spoofing and phishing. To simulate this vulnerability, we set the DMARC policy to "none," which would potentially allow the acceptance of spoofed emails. This condition was tested using the pattern "v=DMARC1; p=none," which matches any DMARC policy that takes no protective action. During our testing, the txt_check function successfully identified this pattern, showcasing the effectiveness of our tool in recognizing and reporting potential vulnerabilities in DNS TXT records.

4.2 Limitations

First, the tool is only so far accurate in assessing the CNAME vulnerability as much as the entries in fingerprint hold. If a sub-domain is pointing to a website that has a custom 404 page whose pattern is not saved in our fingerprint website, then the tool might fail to detect a common message seen with resources being deleted. For example, with content held on GitHub pages, it tells us that “There isn't a

GitHub Pages site here." This allows our tool to use our fingerprint to see if such a text exists on the page where the resource is hosted. By not having an entry in our fingerprint, we will not be able to make matches as we do.

Second, our tool is currently only able to delete the vulnerable subdomain if it is hosted through Cloudflare. This is because we were not able to find APIs for other cloud hosting platforms that would allow our tool to automatically and remotely delete the DNS record. In most cases, the hosting platforms are expecting a user to get rid of it through their dashboard manually. Cloudflare had an API we could use, that is why our tool is limited to deletion to subdomains hosted through Cloudflare only.

Last, due to the nature of subdomain takeover, our tool cannot intrinsically check if a subdomain has been taken over. Therefore, this makes our tool a prevention tool rather than a remedy tool. The tools should be used before you get compromised because after you get compromised, the tool will be deemed useless as the attacker would have successfully hosted content that is pointing from a vulnerable subdomain.

4.3 Demo Display

The demo can be found at this link: Cybersecurity Project Demo.mp4

5. Team Contribution

Name	Wei Chen	Nurassyl Medeu	Jiawei Sun	Jeet Vijaywargi
Contribution	25	25	25	25

References

1. **Github source code for dnsrecon:** https://github.com/darkoperator/dnsrecon
2. **Github source code for Sublist3r:** https://github.com/aboul3la/Sublist3r
3. **Github source for a list of third parties "Fingerprints":** https://github.com/EdOverflow/can-i-take-over-xyz