AUTO – OSINT & VULNERABILITY CHECK

OREOLUWATOMIWA IBIKUNLE

SETU CARLOW

TABLE OF CONTENTS

Abstract

1980s; First cybercrime by Ian Murphy, aka, Captain Zap.

1999; Common Vulnerability and Exposure (CVE) system launched with 1020 published vulnerabilities.

2005; CVEs were being published at a rate of 400 a month.

2010; 40,000 published vulnerabilities

Fast forward over a decade later, Vulnerability management which is the “cyclical practice of identifying, classifying, prioritizing, remediating and mitigating” software vulnerabilities has become the best practice due to such significant rises in the variety of attacks. One of the attacks involves hackers penetrating a whitelisted subdomain to perform nefarious acts on the main application.

VM, traditionally require configuration of scan policies and scheduling and long reports dreaded by most.

- buttons for each scan or all
- tabs for reports, configuration and graph
- user should be able to save their session

AUTO – OSINT & VULNERABILITY CHECK

Regardless of one's qualifications or position, ranging from penetration tester, CISO Manager or freelance hacker, one depends considerably on open-source intelligence (OSINT). It is crucial in mapping out the external attack surface of an organization in a process ultimately known as reconnaissance.

1.1 RECONNAISSANCE

Reconnaissance is an important step in locating and obtaining confidential information. To have an effective system, it must be kept continuously objective-oriented and ensure space for easy maneuvers to allow quick development. Most importantly, it must provide accurate and timely information.

Through reconnaissance, attackers find useful detailed information that open doors for them to directly interact with open ports or running services, even without actively engaging the network.

Contrary to what many might think, hackers do not always attack a company's central server and gain access to the entire network. They gather pieces of information from subdomains, error headers, S3 Buckets, etc and then construct a diagram of the network with the services, ports, and applications inside the environment.

Depending on the type of information needed and the level of authorization needed to get it, recon could take between weeks to months.

One of the first phases is Dumpster Diving which is simply going through someone's bins, digital or physical, to find evidence or information that provides insight into their personal life or confidential information.

Humans are the weakest link in an information security chain and so dumpster diving can be a gold mine to investigators. From old driver's licenses, receipts to digital signatures and private passwords/PINs, anything can be retrieved from dumpster diving.

The phase that comes after most of the time is FootPrinting, which impersonates a website by mirroring it or collects data on the security posture to draw a network map to understand the network infrastructure. It provides important information such as the domain name, TCP and UDP services.

There are 2 main types of reconnaissance:

active – obtaining information about systems using automated scans and manual testing, ping and netcat. It is faster than passive and more accurate but comes with a higher risk of detection. It involves port scanning which identifies open ports and retrieving and analyzing data from them.

passive – gathering information without interacting with the system directly. Like OSINT, it involves gathering information from public resources.

1.2 SUBDOMAIN ENUMERATION

Subdomain enumeration is the process of identifying valid subdomains for a given domain. It is often used to identify less-protected subdomains that are more vulnerable to attack and for recon into an organization's infrastructure.

Vulnerable subdomains can be used to gain access to another more important subdomain or the main domain and to launch social engineering attacks.

Pen Testers can figure out the services an organization offers and most of the information from this is used as part of a security assessment. It helps in finding potential weak points and old, unnecessary or broken-down applications which are not being maintained. Subdomain enumeration also discloses misconfigured DNS entries that reveal sensitive information about a network's internal network structure.

2 AUTOMATION & ORCHESTRATION

Regardless of the industry, automation plays a big part in improving efficiency and removing inherent errors. It helps to save money in the long run and boosts employee morale.

Most times it is used loosely for situations where some parts still need to be manual. Automation, in a nutshell, is the setting up of one task to run on its own. It could be anything from launching a web server to redirecting emails to a predetermined folder.

Most of the time when we speak about automation, we are discussing orchestration which is more complex. It is the automation of many tasks together to form a fully functional process. It involves knowing and understanding the number of steps involved throughout the process. Also, each step needs to be monitored across apps, devices, and databases. (Watts, 2020)

2.1 WHY ORCHESTRATE?

Vulnerability scans note the network's assets within the network from devices to firewalls and constantly collects their operation details. Most scanning tools also come with the ability to audit, log logs, perform threat modelling and remediation that allow organizations to assess fluctuations in their security levels.

Usually, hackers rely on the element of surprise by swiftly swooping in to use application vulnerabilities to enter the system through sites that become live before they are assessed, but with automated vulnerability scanning, teams can detect and fix these vulnerabilities before assets are compromised.

Long term, it reduces costs and development time as the tests are seamlessly performed within the given scope. Money and time are not spent on remediation techniques and employees can perform other duties whilst waiting for a report.

In an environment where employees have less tedious work to do, money and time are saved, everyone is more likely to comply with data processing regulations and as there is little to no error, there is a standard that is upheld according to the General Data Protection Regulation (GDPR), Health Information Privacy and Accountability Act (HIPAA), and the Payment Card Industry Data Security Standard (PCI-DSS).

On the flip side, building a tool that automates full scans can have the opposite effect by clogging CI/CD (continuous integration and deployment) pipelines and overwhelming teams with unnecessary findings. Therefore, only the required tests need to be run, when they are needed, and results need to be filtered based on the level of risk so the important matters can be focused on.

2.2 USING PYTHON FOR TASK AUTOMATION

Python's syntax resembles plain English, making it relatively easy to use. Python comes with great data struct support which enables the user to store and access data such as lists, dictionaries, tuples and sets. These allow efficient management of data and increases software performance. Python also allows the creation of personal data structures which is important on a deeper level.

Almost everything can be automated with Python. From sending emails and filling out PDFs and CSVs (Excel) to interacting with external APIs and sending HTTP requests. It also comes accompanied by many libraries that allow developers to tackle issues relating to machine learning and managing a computers OS.

BeautifulSoup

There are many sites that offer API's and downloadable data sets but there just as many that don't offer these options. Which is why developers sometimes need to extract information from mostly HTML from HTTP responses.

BeautifulSoup is one of the best Python libraries for parsing HTML and XML. It makes web scraping much easier and provides "Pythonic" idioms for iterating, searching, and modifying the parse tree.

SpiderFoot uses this library in a few different modules, including:

2.2.1 SFP_DNSDUMPSTER

The DNSDumpser module performs passive subdomain enumeration using dnsdumpster. BeautifulSoup extracts a CSRF token and uses it to parse the HTML and extract out subdomains from https://dnsdumpster.com :

```
html = BeautifulSoup(str(res2["content"]), features="lxml")

escaped_domain = re.escape(domain)

match_pattern = re.compile(r"^[\w\.-]+\." + escaped_domain + r"$")


for subdomain in html.findAll(text=match_pattern):

    subdomains.add(str(subdomain).strip().lower())
```

2.2.2 SFP_EMAILFORMAT

This module takes a domain name and searches the site https://www.email-format.com/ to find the email address format in use by companies. and it will (hopefully) return email addresses from the given domain. BeautifulSoup is used in this module to parse the response data, extract the table body out to then pass it to our custom parseEmails function.

```
html = BeautifulSoup(res["content"], features="lxml")
if not html:
    return
tbody = html.find('tbody')
if tbody:
    data = str(tbody.contents)
else:
    # fall back to raw page contents
    data = res["content"]
emails = self.sf.parseEmails(data)
```

2.2.3 SFP_REVERSEWHOIS

Reversewhois.io is a search engine used to perform domain enumeration on a company or individual. It takes a domain name to this module and returns any affiliated domains, as well as domain registrar information. BeautifulSoup here parses the response data and iterates through all of the rows in the HTML table to extract out the Reverse Whois information on the provided domain. You can see the full module code here.

```
html = BeautifulSoup(res["content"], features="lxml")
date_regex = re.compile(r'\d{4}-\d{2}-\d{2}')
registrars = set()
domains = set()
for table_row in html.findAll("tr"):
    table_cells = table_row.findAll("td")
    # make double-sure we're in the right table by checking the date field
```

```
    try:
        if date_regex.match(table_cells[2].text.strip()):
            domain = table_cells[1].text.strip().lower()
            registrar = table_cells[-1].text.strip()
            if domain:
                domains.add(domain)
            if registrar:
                registrars.add(registrar)
    except IndexError:
        self.debug(f"Invalid row {table_row}")
        continue
ret = (list(domains), list(registrars))
if not registrars and not domains:
    self.info(f"No ReverseWhois info found for {qry}")
```

2.2.4 REQUEST LIBRARY

Request is an HTTP library in Python that makes it much easier to interact with the web or talk to APIs on twitter or Instagram.

The script below performs a GET request to execute the get-software-information RPC on a remote Junos OS device with the rest API service over port 3000

```
from junos import Junos_Context
import jcs
import requests
```

```
user = Junos_Context['user-context']['user']

password = jcs.get_secret('Enter user password: ')


r = requests.get('http://198.51.100.1:3000/rpc/get-software-information', auth=(user,
password))

print (r.status_code)

if (r.status_code == requests.codes.ok):

    print (r.text)
```

It is used under the Qxf2 automation framework which supports API testing and is much easier than GUI automation.

Get method using Request

```
def get(self, url, headers={}):
        "Get request"
        json_response = None
        error = {}
        try:
            response = requests.get(url=url,headers=headers)
            try:
                json_response = response.json()
            except:
                json_response = None
```

```
        except (HTTPError,URLError) as e:
            error = e
            if isinstance(e,HTTPError):
                error_message = e.read()
                print("\n******\nGET Error: %s %s" %
                    (url, error_message))
            elif (e.reason.args[0] == 10061):
                print("\033[1;31m\nURL open error: Please check if the API
server is up or there is any other issue accessing the URL\033[1;m")
                raise e
            else:
                print(e.reason.args)
                # bubble error back up after printing relevant details
                raise e # We raise error only when unknown errors occurs
(other than HTTP error and url open error 10061)

        return {'response':
response.status_code,'text':response.text,'json_response':json_response,
'error': error}
```

Post method using Request

```
def post(self, url,params=None, data=None,json=None,headers={}):
        "Post request"
        error = {}
        json_response = None
        try:
            response =
requests.post(url,params=params,json=json,headers=headers)
            try:
                json_response = response.json()
            except:
```

```
                json_response = None
        except (HTTPError,URLError) as e:
            error = e
            if isinstance(e,HTTPError,URLError):
                error_message = e.read()
                print("\n******\nPOST Error: %s %s %s" %
                    (url, error_message, str(json)))
            elif (e.reason.args[0] == 10061):
                print("\033[1;31m\nURL open error: Please check if the API
server is up or there is any other issue accessing the URL\033[1;m")
            else:
                print(e.reason.args)
                # bubble error back up after printing relevant details
            raise e

        return {'response':
response.status_code,'text':response.text,'json_response':json_response,
'error': error}
```

Put method in Request

```
def put(self,url,json=None, headers={}):
        "Put request"
        error = {}
        response = False
        try:
            response = requests.put(url,json=json,headers=headers)
            try:
                json_response = response.json()
            except:
                json_response = None
```

```
        except (HTTPError,URLError) as e:
            error = e
            if isinstance(e,HTTPError):
                error_message = e.read()
                print("\n******\nPUT Error: %s %s %s" %
                      (url, error_message, str(data)))
            elif (e.reason.args[0] == 10061):
                print("\033[1;31m\nURL open error: Please check if the API
server is up or there is any other issue accessing the URL\033[1;m")
            else:
                print(str(e.reason.args))
            # bubble error back up after printing relevant details
            raise e

        return {'response':
response.status_code,'text':response.text,'json_response':json_response,
'error': error}
```

Delete method using Request

```
def delete(self, url,headers={}):
        "Delete request"
        response = False
        error = {}
        try:
            response = requests.delete(url,headers = headers)
            try:
                json_response = response.json()
            except:
                json_response = None
```

```
        except (HTTPError,URLError) as e:
            error = e
            if isinstance(e,HTTPError):
                error_message = e.read()
                print("\n******\nPUT Error: %s %s %s" %
                      (url, error_message, str(data)))
            elif (e.reason.args[0] == 10061):
                print("\033[1;31m\nURL open error: Please check if the API
server is up or there is any other issue accessing the URL\033[1;m")
            else:
                print(str(e.reason.args))
            # bubble error back up after printing relevant details
            raise e

        return {'response':
response.status_code,'text':response.text,'json_response':json_response,
'error': error}
```

2.3 SALT

SaltStack, also known as Salt is a fast, scalable and flexible systems management tool written in Python for data center automation, cloud orchestration and configuration management. The basis of its creation was the need for quick and easy communication with thousands of servers in seconds as well as data collection and execution. (Salt, 2022)

It is highly modular and easily extensible to fit the needs of the user. Python ZeroMQ library supports its high-speed tasks and builds up its networking layers.

Authentication and encryption are integral values of Salt, and it uses public API keys for the former and AES encryption for payload communications via msgpack which enables fast and light network traffic.

In terms of its scalability, Salt can be written as plain Python or called from the command line to execute one-off commands or operate as a core part of a larger system. The remote execution architecture of Salt manages diverse commands at high speeds on a single or multiple servers.

2.3.1 Salt's Masters and Minions

Salt utilizes a master-client / publisher-subscriber model in which the salt "daemon", which controls "minions", includes the viable and transparent AMQ broker and leverages state modules (Execution, State, Grains, Renderer, Returners, Runners) that execute the code to setup, enforce or alter the configuration of a system. The minions use "grains" to detect static information (targets) and store it in RAM for fast and easy retrieval.

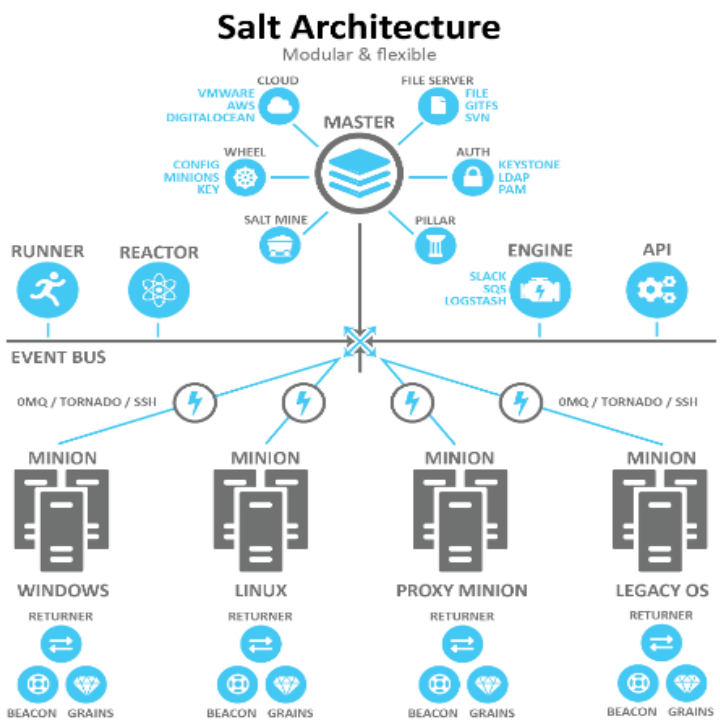


Fig 1: Salt Architecture

An example of a simple command the master could give to minions, is **salt -v '*' pkg.install vim** with the '*' being the target. The target could be other minions targeted by their grains (shared traits). When a minion has successfully executed a job, it sends data back to the master through 2 default ports which work hand in hand to receive and deliver data to ZeroMQ. ZeroMQ is Salt's message bus that creates an asynchronous network topology, using a collection of request-reply and publish-subscribe patterns to provide the quickest code possible. It provides a flexible transport layer alongside Tornado, a full TCP-based transport layer event system. Salt's communication bus is more efficient than a higher-level web service (http) and it allows for decentralized remote execution.

Salt States

The core of the salt state, Salt State File, which makes configuration management possible, is set up with information about the state a system should be in. Salt is very flexible in this area as the more state files are written, the clearer they all become and the more specific to the needs of the developer. Developers also use Salt States to deploy and manage infrastructure with YAML files.

```
apache:
  pkg.installed: []
  service.running:
    - require:
      - pkg: apache
```

This ensures that apache is installed and running. Line 2 and 3 are written in the format <state_module>.<function>.

pkg.installed ensures that the software package, in this case, Apache, is installed by the system's native package manager while service.running keeps the system daemon running.

Another use of States is to automate recursive and predictable tasks by lining up tasks for implementation without the need for manual user input. This boosts scalability alongside the top.sls file which maps salt states to their applicable minions and all these states are run at once by highstate execution.

An example of a top sls file based in the default salt environment(base):

```
# File: /srv/salt/top.sls

base:
  '*':
    - all_server_setup

  '01webserver':
    - web_server_setup
```

The code above basically indicates that a state called all_server_setup should be applied to all '*' minions and web_server_setup should be applied to 01webserver minions.

To run this, the state.highstate function would be used:

salt * state.highstate

Salt Pillar - This feature takes a pre-defined data based on Salt master and distributes it to minions. Its main use is storing secrets and hiding sensitive data or data that is not meant to be directly in the state files.

Beacon – the beacon is a monitoring tool that listens for a system process on salt minions and performs automated reports and error log delivery.

Reactors – They watch salt's event bus for tags that match a given pattern then trigger actions in event response. It could be anything from notifying administrators, to restarting failed applications. Reactors aid with infrastructure scaling and when used with beacons they create unique, personally customized states.

Salt Runner- executes on the salt master and is as simple as a client call.

3 OWASP

The OWASP Top 10 represents a broad consensus and ranking, among security experts, of the most critical web application security risks. Developers keep this standard in mind when performing robust tests and writing secure code for web applications to minimize these risks. Before going in depth into the OWASP top 10 vulnerabilities, the bigger question is, what is OWASP and its mission.

3.1 WHAT IS OWASP

The Open Web Application Security Project (OWASP) is a non-profit organization dedicated to upholding the standards of software security. They ensure ease of access and free use of open-source software development programs, toolkits, and forums under their “open community” model.

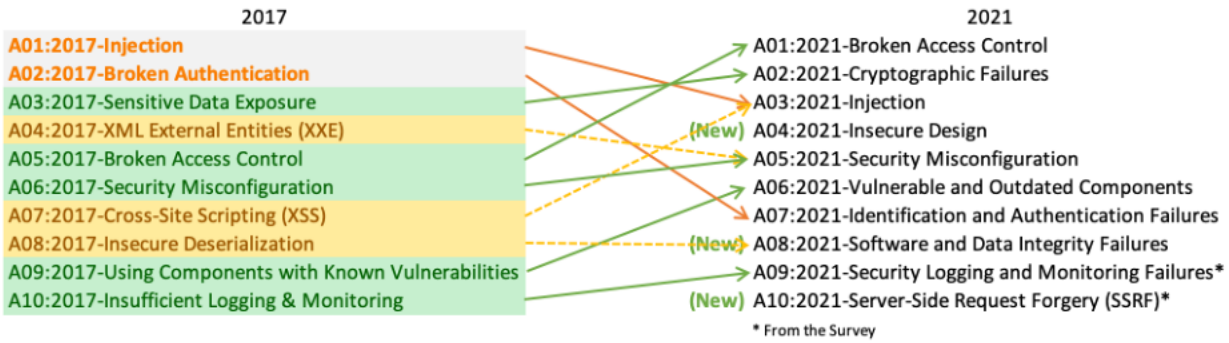
3.2 HOW IS THE OWASP TOP 10 LIST USED AND WHY IS IT IMPORTANT?

OWASP was founded in 2001 but the Top 10 list originated in 2003 and is updated every 3 years. It does more than simply rank vulnerabilities; it also provides remediation advice for the most serious web application security dangers. The risks are ranked according to the severity of the vulnerabilities and how frequently they occur.

In Veracode’s State of Software Security Volume 11, it was found that in a scan of 130,000 applications, nearly 68% of applications had a security flaw under the Top 10. Because they are so common, app developers that take concrete steps to learning them and how to resolve these flaws by integrating it into their software development life cycle (SDLC), show that they have a commitment to the industry’s best practices and are actively trying to keep up. In fact, auditors view an organization’s remiss to address the OWASP Top 10 as an indication that it may be lacking in its compliance standards.

There have been many changes to the style of the list but in 2021, 3 new categories and 4 changes to naming and scoping as well as some consolidations were implemented. The following image depicts this:

c



KEY: → Vulnerabilities promoted in importance

Vulnerabilities → demoted in importance

→ Vulnerabilities removed and merged

3.3 2021 TOP 10

In order of most dangerous to least:

Broken Access Control: moved up from #5 to #1, after OWASP discovered 94% of applications have an access control weakness. Basically, if an attacker uses this exploit to function as a user or as an administrator in the system.

Example: a user might be able to access some resources or perform some action that they are not authorized because the application allows a primary key to be changed.

Mitigation: Developers should deny access by default, excluding public resources and validate JSON Web Tokens after logout. Disable server directory listing and avoid storing sensitive data in root. Fortify and reuse strong access control mechanisms.

All of this can be performed by an Interactive Application Security Testing solution (IAST) such as Seeker. It detects CSRF and pinpoints any bad logic used to handle JSON Web Tokens. Pen Testing supplements its activities as well by helping to detect any unintended access controls.

Cryptographic Failures: moved up from #3 and formerly known as sensitive data exposure which only represented it as a broad symptom, rather than the root cause. The focus has been shifted to cryptography failures when important transmitted data is compromised.

Example: A bank fails to adequately protect its sensitive data by following the PCI DSS to the letter and ends up becoming an easy target for credit card fraud/ identity theft.

Mitigation: Apply appropriate security controls to sensitive data and discard unnecessary sensitive data, using tokenization or truncation. Passwords are a common target and must be stored using salted and peppered hash functions like scrypt or bcrypt. Also, encrypt data at rest and in transit using TLS or HTTP HSTS and caching must be disabled for all these.

Seeker is useful in this situation as it scans for inadequate or weak hardcoded cryptographic keys. Coverity static application security testing (SAST) and Black Duck software composition analysis (SCA), combined with IAST, provide continuous monitoring and verification to ensure sensitive data is not leaked during integrated testing with other internal and external software components.

Injection: demoted from #1 to #3, injection which includes cross-site scripting, occurs when invalid data is sent by an attacker into a web application to perform functions other than what it was designed to do.

Example: An application might use untrusted data when constructing a vulnerable SQL call.

Mitigation: The best mitigation tactics here are to utilize safe API which avoids the use of interpreter entirely and “whitelist” server-side input validation. Also incorporate the use of escape special characters and SQL Controls within queries to prevent mass disclosure of records in case of SQL Injection.

SAST and IAST tools help to identify injection flaws at the static code level and tools like Seeker, help secure the software application.

Insecure Design: newly added vulnerability in 2021 that focuses on design flaw risks. It shines a light on the growing focus on "shifting left" which involves making changes in when, where and how to apply security best practices. So basically, a task that is traditionally done at a later stage of the process and perform that task at earlier stages.

Example: With the traditional Waterfall model, testing is done just before releasing the product into production.

Mitigation: Establish a secure software development lifecycle (SSDLC). Leverage threat modeling to design critical features like authentication and access control. Integrate security concerns and controls into all user stories.

Seeker IAST detects vulnerabilities and exposes all the inbound and outbound API. Weaknesses in the design are made clear by providing a visual map of the data flow and endpoints involved.

Security Misconfiguration: moved up from #6 to #5. 90% of apps tested by OWASP had a design or config weakness that resulted from a configuration error or shortcomings.

Example: A default account and its original password are still enabled, making the system vulnerable to exploit.

Mitigation: Employ a fast and easy hardening process for applications by automating update configurations, apply patches and security advisories regularly. Developers do well to minimalize system setup to eliminate any unnecessary features and components.

Coverity SAST includes a checker that identifies how much information is exposed through an error message. Dynamic tools like Seeker IAST can do this but also detects inappropriate HTTP header configurations during application runtime testing.

Vulnerable and Outdated Components: previously known as "Using Components with Known Vulnerabilities", moved from number #9.

Example: A development team might not have full extensive knowledge of all the components used in their application due to the volume and some expired ones that are vulnerable to attack might get lost in the crowd.

Mitigation: Eliminate unused features and files. Only use components from official signed sources and maintain an inventory of them from both the client and server side using software composition analysis (SCA) tools. Continuously scan for vulnerable components and urgently remediate vulnerabilities or removed compromised components.

Identification and Authentication Failures: renamed from "Broken Authentication", moved down from #2, due to growing use of standard authentication frameworks.

Example: A web application allows the use of weak or easy-to-guess passwords.

Mitigation: Harden all authentication-related processes by employing the use of multi-factor authentication and limiting or delaying failed login attempts.

MFA helps reduce the risk of compromised accounts and auto static analysis aids in finding such flaws whereas manual static analysis adds strength when evaluating custom authentication schemes. Seeker IAST can detect hardcoded credentials and improper authentication.

Finally, use NIST 800-63 B section 5.1.1 for setting up guidelines for Memorized Secrets.

Software and Data Integrity Failures: in response to the huge impact of supply chain attacks, this new category for 2021 entering the list at #8, focuses on the integrity of software updates and CI/CD pipelines.

Example: An application de-serializes attacker-supplied hostile objects, opening itself to vulnerabilities.

Mitigation: Use libraries such as npm from trusted repositories. Create a strong review process for code and configuration changes such as using digital signatures to verify software from expected source has not been altered.

Seeker IAST can detect deserialization flaws and alerts security to insecure redirects or any tampering with token access algorithms.

Security Logging and Monitoring Failures: previously identified as Insufficient Logging & Monitoring. Breaches cannot be detected without logging and monitoring and failures within this category affects visibility, alerting and forensics.

Example: Failure to log events that could be audited, such as logins and failed logins can make an application vulnerable.

Mitigation: Always log login, access control and server-side input validation. Ensure attackers cannot tamper with log data and that they contain enough context to enable in-depth forensic analysis.

Coverity SAST and Seeker IAST can be used to identify unlogged security exceptions.

Server-Side Request Forgery: final new category. It occurs when a web application fetches a remote resource without validating the user-supplied URL. The attacker can then force the app to send a crafted request to any destination regardless of whether the system is protected by a firewall, VPN, or addition network ACL.

Example: Attackers can use connection results or elapsed time to connect to or reject SSRF payload connections to easily map out internal networks and find which ports are open or closed all because the network architecture is unsegmented.

Mitigation: Firstly, disable HTTP redirections and use “deny by default” firewall policies and a positive whitelist with URL schema, port, and destination. Also, isolate remote resource access functionality in a secluded separate network to reduce impact.

Due to its evolved level of agent-technology, Seeker can track, monitor, and detect SSRF, and any potential exploits from it, without the need for additional scanning or triaging. (Synopsys, 2021)

4 MICROSOFT IIS VULNERABILITIES

The MS IIS Default Page Vulnerability is a low-risk vulnerability that is often overlooked and is often found on networks worldwide. Hackers are aware of it and it is usually their point of attack as it is hard for developers to detect and even resolve.

Hackers also leverage IIS extensions as backdoors into servers. IIS backdoors are hard to detect as they are usually opened in directories of legitimate modules, and they follow the same code structure as these modules. Most of the time hackers are persistent and do not start their attack heavily. The first step is is finding an exploitable vulnerability before dropping the script web shell as the first payload. Then the attacker could implement the backdoor method and keep access to the server to monitor traffic and steal credentials.

5 TOOLS

Information about a specific IP address or domain can be found through a variety of services. There are websites like **DSNlytics** and **IPinfo** that provide detailed information about IP addresses, including geolocation, reverse DNS, ASN, related domains, and more. **IIKnowWhatYouDownload** reveals all torrent files associated with an IP address.

A domain name provides a lot of information, such as IP addresses, subdomains, other related domains, registrar details, and contact information.

The certificate database **crt.sh** contains all publicly issued certificates both past and present.

When a URL is submitted, **Urlscan.io** browses it, records the activities that take place during this process (for example, v19 3. OSINT Sources and Tools listed domains and IP addresses), saves the resources of these domains, and takes screenshots.

5.1 EXISTING AUTOMATION TOOLS

OSINT IS A VERY EXTENSIVE TOPIC AND THERE IS SUCH A VAST DIVERSITY OF INFORMATION AVAILABLE THAT THERE ARE NUMEROUS WAYS IN WHICH TO COLLECT OSINT. EACH PROBLEM THAT

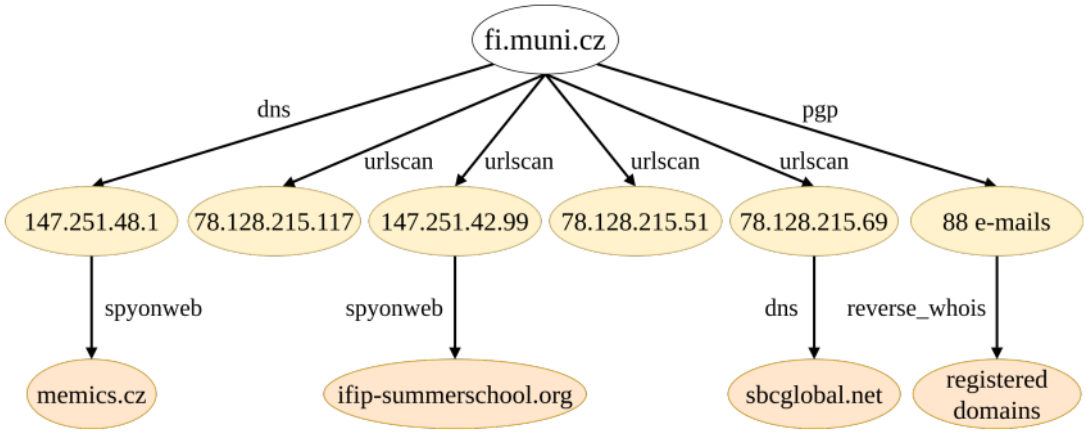
THE USERS OF EXISTING TOOLS MIGHT NEED TO SOLVE REQUIRES A DIFFERENT APPROACH AND HAVING A PERFECT TOOL FOR EACH PROBLEM IS WORTH LESS THAN IT SEEMS.

5.1.1 PANTOMATH

Pantomath is a highly modular framework that provides a complete environment for collecting and evaluating OSINT about IP addresses, emails addresses and domain names. Basically, Pantomath automates the collection of OSINT by employing already existing tools as the number of possible sources is already and high and new ones could appear in the future.

Unfortunately, due to this, Pantomath is directly reliant on these other tools and needs to address any changes that alter its current implementation. On the flip side, it does provide a reliability estimate that does not necessarily provide guarantees of the information provided but rather allows the investigator to make a more informed decision.

Figure 4.1: An illustration of the search tree when querying fi.muni.cz up to a depth of 2.



5.1.2 SEEKER IAST

Referenced in the mitigation area of Section 3.3

6 Conclusion

The research conducted has set the foundation for implementing a tool that automates Open-Source Intelligence and performs vulnerability checks. Selecting any of the forementioned tools may help you to fix security vulnerabilities but not with the same efficiency.

This tool will not just check for some of the most popular vulnerabilities but also the easiest one that is overlooked like IIS Default Page. It will eliminate the manual effort in subdomain enumeration and vulnerability checks thus saving time and reducing human error whilst holding to a certain scalability

BIBLIOGRAPHY

Google. (n.d.). Retrieved from Google Translate: https://translate.google.com/

Intellipaat. (2022, May 5). *reconnaissance-in-cyber-security*. Retrieved from intellipaat.com: https://intellipaat.com/blog/reconnaissance-in-cyber-security/

Kishore, S. (n.d.). *reconnaissance the eagles eye of cybersecurity*. Retrieved from sisainfosec.com: https://www.sisainfosec.com/blogs/reconnaissance-the-eagles-eye-of-cyber-security/

Open Source Intelligence Market. (2022). Retrieved from www.expertmarketresearch.com: https://www.expertmarketresearch.com/reports/open-source-intelligence-market

OWASP Top 10 2021. (2021). Retrieved from Synopsys: https://www.synopsys.com/glossary/what-is-owasp-top-10.html

Owasp Top10. (2022). Retrieved from Imperva: https://www.imperva.com/learn/application-security/owasp-top-10/

Poel, I. v. (2020). Core Values & Value Conflicts in Cybersecurity. *Beyond Privacy vs Security*.

Poel, I. v. (2020). Core Values and Value Conflicts in Cybersecurity. *Beyond Privacy vs Security*.

Rai, B. K. (2020). *Open Source Intelligence Initiating Efficient Investigation and Reliable Web Searching.*

Salt. (2022). Retrieved from Quintagroup: https://quintagroup.com/cms/python/salt

Snell, S. (2009). *Redesigning Craigslist With Focus On Usability*. Retrieved from https://www.smashingmagazine.com/2009/03/redesigning-craigslist-with-focus-on-usability/

Soegaard, M. (n.d.). *Usability: A part of the User Experience*. Retrieved from https://www.interaction-design.org/literature/article/usability-a-part-of-the-user-experience

Spy On Web. (2020, 11 16). Retrieved from https://spyonweb.com/

Synopsys. (2021). *What Is Owasp Top 10*. Retrieved from Synopsys: https://www.synopsys.com/glossary/what-is-owasp-top-10.html

Watts, S. (2020, 09 07). *IT Ochestration vs Automation Whats The Difference*. Retrieved from BMC: https://www.bmc.com/blogs/it-orchestration-vs-automation-whats-the-difference/