Under-Graduate Project

REPORT

Android Penetration Testing

Authors: Chaitanya Bramhapurikar(230305)

Rahul Meena(230832)

Background

- This project explored Android Penetration testing. Several tools and techniques were used to assess and study the android platform and the apps running on it.
- We prepared adapters, scripts to automated workflow and to integrate the tools into a unified UI based platform for penetration testing.

Work

- The main objective was to develop python scripts to automate testing by removing the need to manually analyse and interpret the tools results which can be overwhelming and complicated.
- Here are some of the scripts that were created and tools used throughout the project.

Nmap + Metasploit (metan.py)

- **Functionality**:
    - Metasploit auxiliary modules: They are primarily used for reconnaissance, scanning, and gathering information about a target, rather than directly exploiting vulnerabilities.

    - This Python script automates the process of running a Metasploit auxiliary module against multiple hosts discovered in an Nmap scan.
- **Implementation Methodology**:
    - Leverages Nmap's XML output for target enumeration and service fingerprinting, then determines appropriate Metasploit modules based on port/service signatures.
    - Uses Nmap for a network scan, uses the scan results to select appropriate modules and scanning procedures.
    - Run these modules against the targets discovered in scans and logs the results in json and .txt format.
- **Operational Benefits**:
    - Eliminates repetitive module selection and execution across distributed network segments.
    - The script can be a powerful tool for early discovery and scanning of attack surfaces.
- **Future Development Plans**:
    - Use of parallelism and threading to execute modules in parallel to save time.
    - A full metasploit pipeline that interprets the results of the modules reconnaissance results and automatically selects exploits.

Androwarn (androwarn.py)

- **Functionality**:
    - Performs static bytecode analysis of Android APK files utilizing the Androwarn framework.
- **Implementation Methodology**:
    - Leverages Androwarn's scanning engine to decompile and analyze DEX bytecode, then correlates findings against a heuristic ruleset that flags security anomalies including excessive permission declarations, exposed components, and vulnerable WebView implementations.
- **Operational Benefits**:
    - Serves as an initial reconnaissance phase in Android application security assessment by identifying common attack vectors and implementation flaws..

- **Future Development Plans**:
    - Integration of NLP-based vulnerability pattern recognition instead of static rule based detection mechanisms.

Virus Total (virus_total.py)

- **Functionality**:
    - Scans a APK file for malicious behaviour.
- **Implementation Methodology**:
    - Used the Virus Total API to handle uploading and Scanning.
    - Parsing and summarizing the findings in JSON format.
- **Operational Benefits:**
    - Helps identify malicious Android applications.
    - Easy integration into the Testing platform.
- **Future Development Plans**:
    - Combine with Binary analysis tools to analyse Native libraries present in the application.

Droidlysis (droid.py)

- **Functionality**:
    - Performs static analysis of Android applications through automated bytecode parsing, component extraction, and behavioral pattern recognition using the tool droidLysis.
- **Implementation Methodology**:
    - Executes DroidLysis framework for initial APK analysis, then employs simple pattern-matching algorithms against the extracted bytecode and manifest components.
    - Implements a simple multi-tier security scoring system categorizing findings by severity (critical, high, medium, low) with associated CWE mappings.
    - Serializes analysis results into structured JSON output with detailed metadata, vulnerability indicators, and forensic markers.
- **Operational Benefits**:

    - Accelerates security assessment through automated detection of high-risk code patterns, dangerous permission combinations, and anti-analysis techniques.
- **Future Development Plans**:
    - Integration of machine learning models to replace rule-based detection systems for improved accuracy and reduced false positives.
    - Enhancement of the existing pattern recognition to identify and label more CVEs in the static analysis result.

Automatic APK MITM attack setup (auto_apk_mitm_patch.py)

- **Need and Background:**
    - Inspecting a mobile app's HTTPS traffic using a proxy is probably the easiest way to figure out how it works. However, with the Network Security Configuration introduced in Android 7 and app developers trying to prevent MITM attacks using certificate pinning, getting an app to work with an HTTPS proxy has become quite tedious.
- **Functionality**:
    - This utility automates the APK modification pipeline to enable man-in-the-middle interception with minimal user intervention
- **Implementation Methodology**:
    - decode the APK file using Apktool
    - replace the app's Network Security Configuration to allow user-added certificates
    - modify the source code to disable various certificate pinning implementations
    - encode the patched APK file using Apktool
    - sign the patched APK file using uber-apk-signer
- **Operational Benefits**:
    - Facilitates transparent TLS inspection through proxy tools like Burp Suite for applications otherwise resistant to interception.

- **Future Development Plans**:
    - API integration with traffic analysis platforms for automated request/response capture and analysis, though architectural constraints in the current penetration testing framework that these scripts are made for present implementation challenges.

Fast Hash Cracking with John the Ripper (john.py)

- **Functionality**:
    - Executes multi-phase hash recovery operations against provided cryptographic hashes.
- **Implementation Methodology**:
    - Implements a staged approach utilizing John the Ripper's engine: first deploying dictionary attacks against a repository of high-probability wordlists containing millions of credentials, followed by mask-based brute force attacks for remaining uncracked hashes.
    - Uses the popular John-The-Ripper tool
- **Operational Benefits**:
    - No manual scraping and searching of wordlists and optimizations.
- **Future Development Plans**:
    - Use more sophisticated methods of brute force methods instead of simply going through every combination and masking.

Netdiscover (automate_netdiscover.py)

- **Functionality**:
    - Scans the local network for active hosts using ARP requests to identify IP-MAC address mappings and associated vendor information.
- **Implementation Methodology**:
    - Automates execution of `netdiscover` by dynamically constructing command-line arguments based on user input.

    - Supports multiple scanning modes including auto-discovery, custom IP range scans, and passive listening.
    - Integrates optional flags like fast scan (`-f`) and hardcore mode (`-S`) for deeper scans.
    - Parses and presents network discovery results via subprocess invocation.
- **Operational Benefits**:
    - Provides quick visibility into live devices on a local subnet, helpful in the early reconnaissance phase.
    - Removes need for manual parameter selection and ensures consistent execution.
- **Future Development Plans**:
    - Add result parsing and logging in JSON or CSV format.
    - Use MAC vendor API to tag identified devices with additional context.

DNSMap (automate_dnsmap.py)

- **Functionality**:
    - Performs brute-force DNS enumeration to identify common and hidden subdomains of a given domain using built-in or user-supplied wordlists.
- **Implementation Methodology**:
    - Wraps around `dnsmap` with CLI inputs for domain, custom delay, wordlists, and result output paths.
    - Parses outputs for valid subdomains and their corresponding IPs.
    - Option to save results in regular and CSV formats using `-r` and `-c` flags respectively.
- **Operational Benefits**:
    - Automates tedious subdomain enumeration and provides structured output for later analysis.
    - Reduces dependency on manual DNS enumeration or 3rd party recon services.
- **Future Development Plans**:
    - Integrate multiple subdomain resolvers for validation.

    - Add a visualization module to map discovered subdomains and their corresponding IP ranges.

DNSenum (automate_dnsenum.py)

- **Functionality**:
    - Performs comprehensive DNS enumeration including subdomain discovery, zone transfers, and whois lookups.
- **Implementation Methodology**:
    - Automates `dnsenum` execution with options to enable aggressive scanning, Google scraping, zone transfer attempts, and subdomain recursion.
    - Parses output for A, NS, MX, and TXT records as well as zone transfer results (AXFR).
    - Allows configurable output file paths in XML or text format for integration into the broader reporting pipeline.
- **Operational Benefits**:
    - Offers deeper DNS reconnaissance than brute-force alone by leveraging multiple enumeration techniques.
    - Helpful in identifying exposed internal DNS zones and understanding domain architecture.
- **Future Development Plans**:
    - Integration with `dnsmap` and `massdns` to cross-verify discovered subdomains.
    - Extract and analyze whois data to pivot into netblock mapping.

WafW00f (automate_wafw00f.py)

- **Functionality**:
    - Detects and fingerprints Web Application Firewalls (WAFs) protecting a target domain or IP.
- **Implementation Methodology**:
    - Automates `wafw00f` scanning via subprocess interface, accepts a target domain or IP.

    - Parses response headers and behavior to infer WAF presence using the signature database.
    - Supports batch mode to scan multiple targets concurrently.
- **Operational Benefits**:
    - Helps tailor payloads for further testing by identifying defensive infrastructure early.
    - Reduces time spent in manual testing and signature comparisons.
- **Future Development Plans**:
    - Add a WAF bypass suggestion module based on known rules and evasion patterns.
    - Visual tagging of WAF types (e.g., Cloudflare, F5, AWS WAF) for pentest reports.

Nmap + Fierce (nmap_with_fierce.py)

- **Functionality**:
    - Combines `fierce` for DNS-based reconnaissance and `nmap` for deep network and service scanning based on discovered hosts.
- **Implementation Methodology**:
    - First invokes **`fierce`** to enumerate subdomains and resolve them to IP addresses.
    - Extracts live IPs from the `fierce` output.
    - Then feeds these IPs into **`nmap`** to perform service discovery, version detection, and port scanning.
    - Aggregates results into structured JSON and text formats for review and further automation.
- **Operational Benefits**:
    - Improves targeting by scanning only those IPs that are part of the DNS-reachable infrastructure.
    - Reduces noise and scan time by avoiding blind sweeps across large network ranges.

    - Combines the strength of subdomain brute-forcing with service-level intelligence.
- **Future Development Plans**:
    - Automate reverse DNS mapping for discovered IPs to identify potential shared infrastructure.
    - Include OS fingerprinting and script scanning (`nmap -A`) based on host criticality or WAF detection.
    - Auto-link discovered services to CVE databases for preliminary vulnerability assessment.

LBD (automate_lbd.py)

- **Functionality**:
    - Detects load balancing and DNS-based distribution mechanisms for a given domain.
- **Implementation Methodology**:
    - Wraps `lbd` to detect multiple IPs for a single host via DNS lookups and response header variations.
    - Checks for round-robin DNS, CDN usage, or HTTP load balancing mechanisms.
    - Outputs binary flags (load balanced: yes/no) with optional verbose mode showing IP list.
- **Operational Benefits**:
    - Useful for determining if multiple probes are hitting different servers, which affects test repeatability.
    - Essential for understanding how a system scales and routes incoming traffic.
- **Future Development Plans**:
    - Integrate with Burp Suite to compare HTTP responses across nodes.
    - Add support for tracing CDN layers like Akamai, Cloudflare, or Fastly.

Masscan (automate_masscan.py)

- **Functionality**:
    - Performs ultra-fast port scanning over large IP ranges with results similar to `nmap`.
- **Implementation Methodology**:
    - Automates `masscan` with flags for target IP range, ports, rate limiting, and output format.
    - Supports JSON and grepable output for rapid post-processing and filtering.
    - Optional argument parsing for stealthier scans (e.g., low rate, randomization).
- **Operational Benefits**:
    - Ideal for fast asset discovery across massive IP blocks.
    - Significantly reduces scan time compared to `nmap` while maintaining reasonable accuracy.
- **Future Development Plans**:
    - Integrate `masscan` with `nmap` service detection for a hybrid quick-deep scanning pipeline.
    - Add scheduling and throttling to bypass rate limits and IDS/IPS detection.

Future Development Ideas

- During the project we realised that “Penetration Testing is all about filtering out information”.
- We strongly believe it since a typical android app can have hundreds of thousands of lines of code and a successful testing session will identify the most interesting ones from it. So in one way it is about filtering large data. Most of the bad security practices and vulnerabilities follow common patterns and behave similarly. This suggests the potential of LLMs and Machine Learning models in this domain.
- We would love to explore it further, since it is an ongoing research area and no tools exist that combine Machine Learning with Security testing.