Application Fingerprinting with Kitsune

Peter Vicherek*, Mark Shtern†
Seneca College School of Applied Arts and Technology
Toronto, Ontario M3J 3M6
Email: *cvicherek@myseneca.ca, †mark.shtern@senecacollege.ca

***Abstract*—The cost of data breaches is rising year by year, at the same time several data breaches occurred due to unpatched/unprotected systems. One of the reasons for a large number of unpatched systems is that IT departments have insufficient knowledge of their vulnerable systems. This paper presents an open source application fingerprinting tool. The tool is capable of pinpointing software and their versions installed on a system. We conducted a case study and successfully discovered vulnerable frameworks in newly developed software systems.**

***Keywords*—web application fingerprinting, version detection, source code analysis**

I. Introduction

The cost of data breaches is rising year by year. According to the Ponemon Institute Data Breach Study, in the last two years the total cost of a breach has increased 23 percent [1] and the average total cost of a data breach reached $3.7 million. These breaches exposed sensitive user data; real names, social security numbers, home addresses, financial information, medical records, passwords and insurance information [2] and the average cost per lost or stolen record has risen to $154 [3]. Despite the fact that the majority of institutions have deployed security measures for data breach prevention, it was reported that there were 472 breaches in the year 2015 alone, and over 139,278,685 records stolen [4]. This indicates that existing security measures are insufficient for data breach prevention.

Several breaches occurred due to unpatched/unprotected systems. Moreover, 99.9% of exploited vulnerabilities were compromised more than a year after the initial CVE was published, based on the Verizon 2015 Report [5]. Evidently, vulnerability patching[1] is a major problem [6] in the industry today.

One of the reasons for unpatched/unprotected systems is that IT teams are not aware of the vulnerable systems on their infrastructure. Several factors contribute to the inability of IT departments to efficiently track running assets in their infrastructure: (1) agility of both infrastructure (e.g cloud, software defined networks, etc) and the business process; (2) increasing complexity of business processes which are often composed of several third party services or are dependent on third party libraries/frameworks; (3) in house developers often do not have a complete picture of their systems [7]. IT departments function as computer power providers (e.g cloud and hosting providers) and are unaware of the applications hosted on their infrastructure.

Since companies often cannot patch their systems because they do not know what to patch, security providers are beginning to develop application-aware security measures. For example, network intrusion detection systems may have the capability to dynamically discover running applications based on network traffic analysis [8]. In addition, several tools such as p0f[2] and BlindElephant[3] were primarily developed to detect running application versions. However, these tools attempt to identify software versions based on the application's run-time behaviour. We propose a novel tool to identify application versions based on static analysis of application assets such as source code, binaries, pictures etc.

In addition to version detection capabilities, the tool is able to identify both the name of the third party framework/libraries and the version of the library for deployed applications. This capability allows for the discovery of vulnerable frameworks. More importantly, the proposed method is able to identify the name/version of applications deployed on a system without prior knowledge of their location. This feature is especially valuable for auditing purposes and gives an auditor the ability to list deployed application on a production system.

The application discovery and version identification process utilizes Reverse Engineering techniques. The process consists of 3 phases: scanning a file storage to collect system artifacts, fingerprinting discovered artifacts, and using a probabilistic model to identify both the application name and its corresponding version. This proposed approach has been implemented in our tool called Kitsune[4].

For an experiment, Kitsune was configured to match against 617 versions across 11 web applications and has accurately identified application versions without prior knowledge of their location on the system. Furthermore it has discovered applications with critical vulnerabilities as presented in our case study. Kitsune is open source[5] and freely available for anyone to download, use, copy or modify. Our experiment has demonstrated both the usefulness and usability of Kitsune.

II. Requirements

Kitsune was designed to meet these requirements.

Open Source: We are strong believers in open source software and the benefits that come with it. We have made the source code freely available for anyone to download, use, modify or copy without restrictions.

Ease of Use: We believe Kitsune should be easy to use whether you are a developer or a non-experienced user. By

[1]When we speak about patching we refer to both regular patching and virtual patching (i.e using security measures for protection of vulnerable systems).

[2]http://lcamtuf.coredump.cx/p0f3/

[3]http://blindelephant.sourceforge.net/

[4]Kitsune (*kit soo nay*) - A Japanese mythological creature know for its intellect.

[5]https://github.com/sikula/kitsune

making the interface simple yet flexible we minimize the time it takes to learn a new tool as well as avoid the cost of hiring specialized people to use the tool.

Multiple Formats: To make Kitsune friendly with other applications, we support various output formats including: JSON, JSON (simplified), CSV, YAML, ASCII-Table and HTML. Further more we give users the flexibility to write their own formats to best suit their needs.

III. Methodology

Unlike other fingerprinting tools such as BlindElephant which analyzes files in known places or Snort[6] which analyzes network traffic, Kitsune accurately determines versions by analyzing the source code of files on a system. This method takes advantage of all system artifacts such as source code files, binaries, pictures, etc.

Kitsune recursively scans a specified directory to produce an Artifact, a set of fingerprints. The Artifact is put through our matching algorithm which cross references the Artifact with a repository of known fingerprints stored in a database repository. The output of the matching algorithm is then placed through a Probabilistic Model to determine versions of applications on the system. This concept is illustrated in Figure 1. The Probabilistic Model is defined as followed:

$$\text{Probability} = \frac{\text{Matched Fingerprints}}{\text{Total Fingerprints}}$$

with the restriction

$$0.0 \geq \text{Probability} \leq 1.0$$

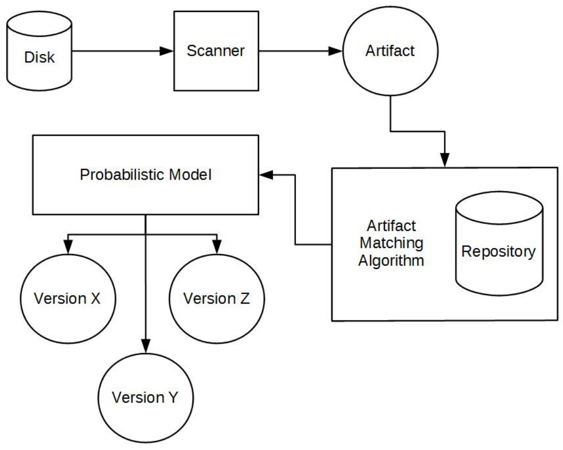


Fig. 1. Illustrates how Kitsune determines application versions

IV. Implementation

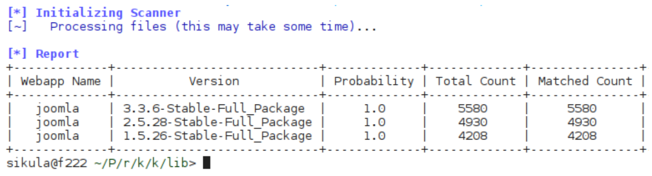
Kitsune is written in Ruby and uses a SQLite database for storing the application checksums. The core is written in 700 lines of code and external assets (javascript, html, css and template files) consuming 9,000 lines of code. The database is roughly 250M in size.

While developing Kitsune, we discovered a bottle neck in our scanner implementation causing us to search for a more optimized hashing algorithm. This led us to *xxhash* a "extremely fast non-cryptographic hash algorithm" [9]. Since Kitsune has no immediate need for a secure hash algorithm xxhash was a perfect choice.

V. Case Study

A. Experiment 1

Our first experiment consisted of determining applications on the system which we already knew the versions to in order to test the accuracy of Kitsune. In this case we have 3 versions of Joomla! installed. [Figure 2]



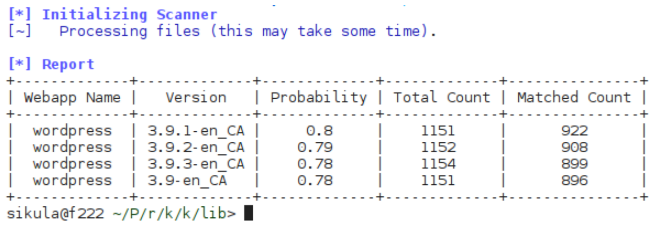
```
[*] Initializing Scanner
[~]   Processing files (this may take some time)...

[*] Report
+-------------+-----------------------------+-------------+-------------+---------------+
| Webapp Name |           Version           | Probability | Total Count | Matched Count |
+-------------+-----------------------------+-------------+-------------+---------------+
|   joomla    | 3.3.6-Stable-Full_Package   |     1.0     |    5580     |     5580      |
|   joomla    | 2.5.28-Stable-Full_Package  |     1.0     |    4930     |     4930      |
|   joomla    | 1.5.26-Stable-Full_Package  |     1.0     |    4208     |     4208      |
+-------------+-----------------------------+-------------+-------------+---------------+
sikula@f222 ~/P/r/k/k/lib> █
```

Fig. 2. Kitsune identified 3 versions of Joomla! installed with near perfect accuracy

B. Experiment 2

Our second experiment consisted of finding open source applications which were built on top of another application. In this example, we looked for applications built on top of Wordpress. We downloaded the newest version of an application we found and Kitsune determined it was using Wordpress 3.9.1 as a base (the newest version of Wordpress being 4.2.3) [Figure 3].



```
[*] Initializing Scanner
[~]   Processing files (this may take some time).

[*] Report
+-------------+-------------+-------------+-------------+---------------+
| Webapp Name |   Version   | Probability | Total Count | Matched Count |
+-------------+-------------+-------------+-------------+---------------+
|  wordpress  | 3.9.1-en_CA |     0.8     |    1151     |      922      |
|  wordpress  | 3.9.2-en_CA |    0.79     |    1152     |      908      |
|  wordpress  | 3.9.3-en_CA |    0.78     |    1154     |      899      |
|  wordpress  | 3.9-en_CA   |    0.78     |    1151     |      896      |
+-------------+-------------+-------------+-------------+---------------+
sikula@f222 ~/P/r/k/k/lib> █
```

Fig. 3. Kitsune showing an application matched wordpress 3.9.1-en_CA with 80% probability

Wordpress 3.9.1 has over 15 vulnerabilities ranging from a severity level of 2.1 up to 7.5. These exploits include: server-side request forgery, cross-site scripting, weak hash implementations, denial of service attacks, protection bypasses and code injection attacks.

VI. Conclusion

We have proposed a novel method for application and version fingerprinting based on the analysis of system artifacts. Our method is capable of detecting both application and versions and any external libraries the application is dependent on. We have implemented our proposed method in our tool, Kitsune. We have conducted a set of experiments using

[6]https://www.snort.org/

Kitsune and discovered multiple vulnerabilities in an open source project. Kitsune is open source and freely available to download from https://github.com/sikula/kitsune.

References

[1] P. I. LLC, *2015 Cost of Data Breach Study: Global Analysis*, 2015. [Online]. Available: https://www14.software.ibm.com/webapp/iwm/web/signup.do?source=ibm-WW_Security_Services&S_PKG=ov34982&S_TACT=C405001W

[2] Symantec, *Internet Security Threat Report*, 2015. [Online]. Available: http://www.symantec.com/security_response/publications/threatreport.jsp

[3] ——, *Internet Security Threat Report*, 2015. [Online]. Available: http://www.symantec.com/security_response/publications/threatreport.jsp

[4] I. T. R. Center, *Data Breach Reports*, 1st ed., 2015. [Online]. Available: http://www.idtheftcenter.org/images/breach/DataBreachReports_2015.pdf

[5] Verizon, *Data Breach Investigation Report*, 2015. [Online]. Available: http://www.verizonenterprise.com/DBIR/2015/

[6] ——, *Data Breach Investigation Report*, 2015. [Online]. Available: http://www.verizonenterprise.com/DBIR/2015/

[7] M. Shtern and V. Tzerpos, *Lossless Comparison of Nested Software Decompositions*, 2015. [Online]. Available: http://www.cse.yorku.ca/~bil/publications/upmojo.pdf

[8] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer, *Dynamic Application-Layer Protocol Analysis for Network Intrusion Detection*, 2015. [Online]. Available: http://www.icir.org/robin/papers/usenix06/

[9] Code.google.com, “xxhash - extremely fast non-cryptographic hash algorithm - google project hosting,” 2015. [Online]. Available: https://code.google.com/p/xxhash/