Xprobe2++: Low Volume Remote Network Information Gathering Tool

Fedor V. Yarochkin, Ofir Arkin *, Meder Kydyraliev †, Shih-Yao Dai, Yennun Huang ‡, Sy-Yen Kuo
Department of Electrical Engineering
National Taiwan University
No. 1, Sec. 4, Roosvelt Road, Taipei, 10617 Taiwan
E-mail: sykuo@cc.ee.ntu.edu.tw

Abstract

Active operating system fingerprinting is the process of actively determining a target network system's underlying operating system type and characteristics by probing the target system network stack with specifically crafted packets and analyzing received response. Identifying the underlying operating system of a network host is an important characteristic that can be used to complement network inventory processes, intrusion detection system discovery mechanisms, security network scanners, vulnerability analysis systems and other security tools that need to evaluate vulnerabilities on remote network systems.

During recent years there was a number of publications featuring techniques that aim to confuse or defeat remote network fingerprinting probes.

In this paper we present a new version Xprobe2, the network mapping and active operating system fingerprinting tool with improved probing process, which deals with most of the defeating techniques, discussed in recent literature.

Keywords: *network scanning, system fingerprinting, network discovery*

Submission Category: *Software Demonstration*

1. Introduction

One of the effective techniques of analyzing intrusion alerts from Intrusion Detection Systems (IDS) is to reconstruct attacks based on attack prerequisites. [8]. The success rate of exploiting many security vulnerabilities is heavily dependent on type and version of underlying software, running on attacked system and is one of the basic required components of the attack prerequisite. When such information is not directly available, the Intrusion Detection System correlation engine, in order to verify whether attack was successful, needs to make "educated guess" on possible type and version of software used at attacked systems.

For example, if Intrusion Detection system captured network payload and matched it to the exploit of Windows system vulnerability, the risk of such detected attack would be high only if target system exists, indeed is running Windows Operating System and exposes the vulnerable service.

In this paper we propose a new version of the Xprobe2 tool[1] (named Xprobe2++) that is designed to collect such information from remote network systems without having any privileged access to them. The original Xprobe2 tool was developed based on number of research works in the field of remote network discovery[12], [3], [1] and includes some advanced features such as use of normalized network packets for system fingerprinting, "fuzzy" signature matching engine, modular architecture with fingerprinting plugins and so on.

The Xprobe2++ basic functionality principles are similar to the earlier version of the tool: the Xprobe2++ utilizes similar remote system software fingerprinting techniques. However the tool includes a number of improvements to the signature engine and fuzzy signature matching process. Additionally, the new version of the tool includes a number of significant enhancements, such as use of test information gain weighting, originally proposed in [4]. The network traffic overhead minimization algorithm uses the test weights to re-order network probes and optimize module execution sequence. The new version of the tool tool also includes modules to perform target system probing at the application layer. This makes the tool capable of successfully identifying the target system even when protocol scrubbers (such as PF on OpenBSD system) are in front of the probed system and normalize network packets[2][5].

Use of Honeynet software (such as honeyd) is also known to confuse remote network fingerprinting. These Honeynet systems are typically configured to mimic actual network systems and respond to fingerprinting with packets that match certain OS stack signatures[9]. Xprobe2++ includes the analytical module that attempts to detect and

*Ofir Arkin is with Insightix
†Meder Kydyraliev is with Google
‡Yennun Huang is with Vee Telecom

identify possible Honeynet systems among the scanned hosts.

This paper's primary contribution is introduction of remote network fingerprinting tool that uses both network layer and application layer fingerprints to collect target system information and is capable of feeding such data (in form of XML) to information consumers (such as Intrusion Detection System correlation engine).

The rest of this paper is organized as follows: Section 2 introduces basic concepts of network fingerprinting and the problems that the tool has to deal these days, and also proposed solutions. Section 3 introduces basic Xprobe2/Xprobe2++ architecture. Section 4 introduces improvements that were brought in Xprobe2++. 5 Section demonstrates some evaluation results and Section 6 discusses possible problems and Section 7 concludes this work.

2. Preliminaries

Network Scanning is the process of sending one or a number of network packets to a host or a network, and based on received response (or lack of such) justifying the existence of the network or the host within target IP address range.

Remote Operating System Fingerprinting is the process of identifying characteristics of the software (such as Operating System type, version, patch-level, installed software, and possibly - more detailed information), which runs on remote computer system. This can be done by analyzing network traffic to and from the remote system, or by sending requests to remote system and analyzing the responses.

The passive analysis of network traffic is frequently named in literature as `passive fingerprinting` and active probing of remote systems is named as `active fingerprinting`.

Xprobe2++ is a novel active remote operating system fingerprinting tool that uses TCP/IP model networking layer protocols and application layer requests to identify the type and version of operating system software, running on target system.

With introduction of application layer tests Xprobe2++ aims at resolving the problems, which can not be resolved by fingerprinting at network layer. In the remaining part of this section we are going to discuss typical problems and issues that a network layer operating system fingerprinting tools have to deal with during the scanning process.

2.1. Modern Fingerprinting Problems

Honeypot systems, modified TCP/IP stack settings and network packet scrubbers are known to frequently confuse remote fingerprinting tools. Honeypot systems often respond as hosts or a group of hosts to remote fingerprinting tools. Modified TCP/IP stack responses are hard to fingerprint with strict signature matching. When packets traverse across the network, they can be modified by network traffic normalizers. All of these factors affect the accuracy of the OS fingerprinting.

Xprobe2++ is aware of these problems and deals with them by using fuzzy matching and mixed signatures that probe target system at different layers of OSI Model network stack.

Moreover, such behavior of some routing and packet filtering devices could be analyzed and signatures to identify and fingerprint intermediate nodes could be constructed.

For example, OpenBSD PF filter is known to return different values in TTL field, when a system behind the filter is accessed [6]. A signature can be constructed to detect this behavior.

3. Tool Architecture Overview

The Xprobe2++ tool architecture includes several key components: core engine, signature matcher, and an extendable set of pluggable modules (also known as plugins). The core engine is responsible for basic data management, signature management, modules selection, module loading and probe execution. The signature matcher is responsible for result analysis. The plugins provide the tool with packet probes to be sent to the target systems and methods of analyzing and matching the received responses to the signature entries.

The Xprobe2++ modules are organized in several groups: Network Discovery Modules, Service Mapping Modules, Operating System Fingerprinting Modules and Information Collection Modules.

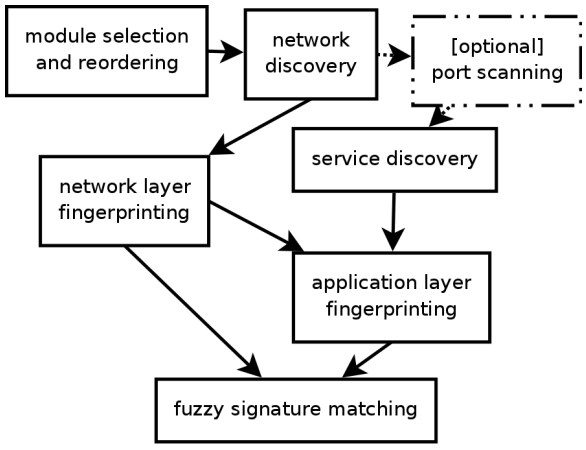

Figure 1. Implementation Diagram

The general sequence of module execution is denoted on Figure 1. Each group of the modules is dependent on successful execution of the other group, therefore groups of modules are executed sequentially. However each particular module within the group may be executed in parallel with another module within the same group.

It is possible to control which modules, and in what sequence are to be executed, using command line switches[1].

3.1. Network Discovery Modules

Xprobe2 discovery modules are designed to perform host probing, firewall detection, and provide information for the automatic receive-timeout calculation mechanism. Xprobe2++ comes with a new module that uses SCTP protocol for remote system probing.

The aim of all network discovery modules is to elicit a response from a targeted host, either a SYN—ACK or a RST as a response for the TCP ping discovery module and an ICMP Port Unreachable as a response for the UDP ping discovery module or an SCTP response for SCTP ping module. The round trip time, which can be calculated for any successful run of a discovery module, is remembered by module executor and is further used by the receive-timeout calculation mechanism. The receive-timeout calculation mechanism is used at the later stage of the scanning to to estimate actual target system response time and identify silently dropped packets without having to wait longer.

3.2. OS Fingerprinting Modules

The Operating System Fingerprinting Modules in Xprobe2++ include both network layer fingerprinting modules that operate with network packets and application layer fingerprinting modules that operate with application requests.

The OS fingerprinting modules provide set of tests for a target (with possible results, stored in signature files) to determine the target operating system and the target architecture details based on received responses.

The execution sequence and the number of executed operating system fingerprinting modules can be controlled manually or be selected automatically based on the information discovered by network discovery modules or provided by command line switches.

3.3. Fuzzy Signature Matching Mechanism

The Xprobe2 tool stores OS stack fingerprints in form of signatures for each operating system. Each signature will contain data regarding issued tests and possible responses that may identify the underlying software of target system.

Xprobe2/Xprobe2++ signatures are presented in human-readable format and are easily extendable. Moreover,the signatures for different hosts may have variable number of signature items (signatures for different tests) presented within the signature entry. This allows the tool to maintain as much as possible information on different target platforms without need to re-test the whole signature set for the full set of fingerprinting modules every time, when the system is extended with new fingerprinting modules.

Following example depicts the Xprobe2++ signature for Apple Mac OS operating system with application layer signature entry for SNMP protocol.

```
fingerprint {
  OS_ID = "Apple Mac OS X 10.2.3"
  icmp_echo_reply = y
  icmp_echo_code = !0
  . . .
  snmp_sysdescr = Darwin Kernel Version
  http_caseinsensitive  = y
}
```

The signature contains the pairs of key, values for fingerprinting tests (key) and matching results (values). The keywords are defined by each module separately and registered within Xprobe2 signature parser run-time.

Xprobe2 is the first breed of remote OS fingerprinting tools that introduced "fuzzy" matching algorithm for the Remote Operating System Fingerprinting process. The "fuzzy" matching is used to avoid impact on the accuracy of fingerprinting by failed tests and the tests, which were confused by modified TCP/IP stacks and network protocol scrubbers. Thus in case if no full signature match is found in target system responses, Xprobe2 provides a best effort match between the results received from fingerprinting probes against a targeted system to the signature database. The details of Xprobe2 "fuzzy" matching algorithm can be found in our earlier publication[1].

In Xprobe2++ the "fuzzy" matching algorithm is updated, so module weights and reliability metrics are used in final score calculation. The original algorithm for module weight calculation is proposed in[4]. Reliability metric is a floating point value in range[0,1], which can be optionally included as part of signature for each test.

4. Tool Improvements

4.1. Application Layer Signatures

Some TCP/IP network stacks may be modified deliberately to confuse remote Operating System Fingerprinting

[1]Please refer to the Xprobe2++ manual page for the detailed information.

attempts. In other cases a network system may simply forward a TCP port of an application. The modern OS fingerprinting tool has to have possibilities to deal with this type of systems and possibly identify the fact of OS stack modification or port forwarding. Xprobe2++ deals with the fact by using additional application layer differentiative tests to map different classes of operating systems. The methods of application layer fingerprinting are known to be effective[2] and it is much harder to emulate application layer responses to match signatures of a particular operating system. The application layer responses are not modified by network protocol scrubbers and thus may provide more accurate information. We do not claim that it is impossible to alter system responses at application layer, but we simply point out there is less motivation to modify system responses at application layer, as this is much more complex task with higher risks of bringing system instability or introducing security vulnerabilities in the application.

The applications running on different operating systems may respond differently to certain type of requests. This behavior is dictated by operating system limitations or differences in design of underlying operating system components. A simple test that verifies 'directory separator' mapping simply tests how target system handles '/' and '\\' type requests. The application will respond differently under Windows and Unix because of the difference in the filesystem implementation. Modifying Application layer responses to respond as other type of operating system is not an easy task. For example, normalization of responses to "..\..\ requests on web server running on the top of OS/2 platform may "unplug" a security hole on this operating system[7].

Xprobe2++ uses application-layer modules in order to detect and correct possible mistakes of fingerprinting at network layer. These modules can also collect additional information on target host. In addition to that, the new version of Xprobe2++ comes with a module that attempts to detect honeyd instances and other "honeypot" systems by generating known-to-be valid and invalid application requests and validating responses. The variable parts of these requests, such as filenames, usernames and so on, are randomly generated to increase complexity of creating "fake" services without full implementation of the application or protocol. Inconsistencies with received application responses are considered as signs of possible honeypot system.

In addition to that, the inconsistency of the results returned by application layer tests and network layer tests may signify presence of a honeypot system, a network-layer packet normalizer or a system running static port address translated (PAT) services.

The detailed list of implemented application layer tests is shown in Table 4.1. As it can be observed from this table, some of these application layer tests can only differentiate between classes of operating systems, while others may identify certain characteristics, such as used filesystem type, which are specific to the particular operating system(s) and and may give some clues of used software version.

We would like to further discuss the groups of application layer tests, which are supported by our tool. However it should be understood that the testing possibility at application layer is not limited by those methods discussed in this section. More specific application layer tests, such as used for HTTP Server fingerprinting [10] or Ajax Fingerprinting Techniques[11] can be used to gain additional precision in remote system fingerprinting process.

`Underlying Filesystem tests` - this group of tests aims at detecting how underlying OS system calls handle various characteristics of directory or file name. For example, FAT32 and NTFS filesystems threat MS-DOS file names, such as FOO~1.HTM, in a special way, file names are case insensitive, requests to file names containing special character 0x1a (EOF marker) will return different HTTP responses from a web server running on the top of Windows (403) and Unix OS (404). `Presence of special files` - This method is not as reliable as filesystem based methods, however it often produces useful results. There are special files on some filesystems, such as Thumbs.db that is automatically created on Windows systems when folder is accessed by Explorer. The file format is different on different OS versions. If such file is obtained, it is possible to validate whether the file was created at the system where it is presently located by comparing the application and the file time stamps.

We also believe it might be possible to perform further differentiation of operating systems at application layer by analyzing encoding types, supported by application or underlying file system. It may also be possible to analyze distribution of application layer response delays for different requests in order to identify "fake" services or fingerprint particular software versions. Further research in this area is needed.

4.2. Optional TCP Port Scanning

One of the motivations for developing the original Xprobe2 tool was to avoid dependency on network fingerprinting tests that would require excessive amount of network probes in order to collect the preliminary information. Xprobe2++ network layer tests are primarily based on variety of ICMP protocol tests. Such tests do not require any additional information of target system, such as UDP or TCP open or closed port numbers simply because there is no "port" concept in context of the protocol.

The optional TCP/UDP port scanning module, when enabled, allows execution of TCP, UDP and application layer tests, because only these tests require knowledge of TCP

Test type	Usable Protocol	Test precision
Directory Separator	HTTP	Windows vs. Unix
New line characters	HTTP	Windows vs. Unix
Special/reserved filenames	HTTP	Windows vs. Unix
Root directory	FTP	Windows,Unix,Symbian,OS/2
Special characters (EOF,EOL		
Filesystem limitations	HTTP, FTP	Correlates FS-type to OS
Filesystem illegal characters	HTTP, FTP	Correlates FS-type to OS
Case sensitivity	HTTP, FTP	Windows vs. Unix
Special filenames handling	HTTP, FTP	Windows vs. Unix
Special files in directory	HTTP, FTP	Windows types, MacOS, Unix
Binary file fingerprinting	FTP	Windows, Unix types

Figure 2. Xprobe2++ Application Layer Tests

and UDP port status.

If optional TCP/UDP port scanning module is not executed, which is default behavior, Xprobe2++ will only use information provided by command line (such as open port numbers), and the ports, which statuses are discovered during execution of other tests. Modules are reordered prior the execution in order to minimize total number of packets and optimize useablity of information that could be discovered during each module execution. For example, the application layer test that uses UDP packet with SNMP query will be placed for execution before the module that requires a closed UDP port. When the SNMP query is sent, the received response (if any) will reveal the status of SNMP port at target system. If the UDP port is closed, the ICMP *Port Unreachable* response would be received. In this case the received datagram is passed to the module that requires closed UDP port. If a UDP packet response is received, the SNMP signatures can be matched to the received response. If no response is received, the result of this test is not counted.

This way Xprobe2++ maintains its minimal usage of packets for the network discovery.

5. Evaluations

We evaluated the new version Xprobe2++ system by executing Xprobe2++ and nmap scans against a number of different network systems: computer hosts, running Linux and windows operating systems and variety of protocols, routers and networked printers. Additionally, we tested Xprobe2++ against a web server system running on Linux operating system and protected by OpenBSD packet filter with packet normalization turned on. We verified correctness of each execution and corrected the signatures, when it was necessary.

The HTTP application module was manually loaded in Xprobe2++ by specifying port 80 as open port in Xprobe2++ command line. The same parameter was passed to Nmap tool. Nmap used port module for TCP ping probe to identify responsiveness of remote system.

We also performed a few test runs by simultaneously executing Xprobe2++ and nmap against unknown network systems and recording network traffic load generated by each tool. The the sampled network traffic throughput, recorded with ntop, is shown on Figure 3. Please note that nmap needs to execute port scanning in order to be able to successfully guess remote operating system type, while Xprobe2++ can rely on results of the tests, which do not require any ports to be known, with exception for application layer module. The diagram simply demonstrate that it is possible to decrease network overhead when no TCP port scanning is performed.

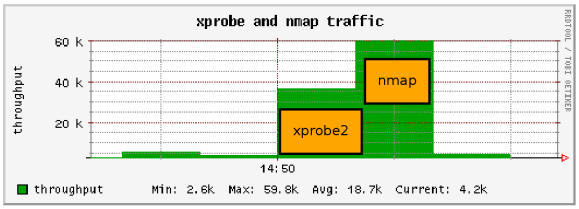

Figure 3. Xprobe2++ and nmap generated traffic loads.

6. Discussions

Our tool provides a high performance, high accuracy network scanning and network discovery techniques that allow users to collect additional information of scanned environment. Xprobe2++ is focused on using minimal amount of

packets in order to perform active operating system fingerprinting, that makes the tool suitable for larger-scale network discovery scans. However these benefits also lead to some limitations, which we would like to discuss in this section.

In order to successfully fingerprint target system, Xprobe2++ needs the remote host to respond to at least some of the tests. If no preliminary information is collected before the tests and some of the protocols (such as ICMP) are blocked, Xprobe2++ results may be extremely imprecise or the tool may actually fail to collect any information at all. We consider this as the major limitation of the tool.

The other limitation with the application-layer tests is that currently Xprobe2++ does not perform network service fingerprinting. By doing so we minimize network traffic overhead and risk of remote service to crash, however Xprobe2++ may also run wrong tests on the services, that are running on non-standard ports or even miss the services, which are running on non-common port numbers. Methods of low-overhead, risk-free network service fingerprinting could be subject of our further research that could resolve this limitation.

Also, despite of the fact that the the tool is capable of performing remote host fingerprinting without performing any preliminary port scanning of the target system, this may lead to significant performance drops when running application-layer tests on filtered port numbers. We believe that preliminary port probe for each application-layer test may be helpful to resolve this limitation.

Xprobe2++ uses libpcap library for its network traffic capture needs. The library provides unform interface to network capture facilities of different platforms and great portability, however it also makes the tool unsuitable for high-performance, large volume parallel network fingerprinting tasks, due to high packet drop ratio on heavily loaded networks. Use of PF_RING sockets, available on Linux platform, may be considered in future releases of this tool in order sacrifice portability for performance improvements.

7. Conclusion

Our primary contribution is demonstration of the tool that is capable of using the application layer fingerprinting tests along with network layer fingerprinting to perform OS fingerprinting remotely with higher precision and lower network overhead. Additionally, the tool can demonstrate that with the use of application layer tests it is possible to detect specific network configurations, which could not be identified by using network layer fingerprinting tests alone.

8. Availability

Developed application is free software, released under GNU General Public License. The discussed version of this software will be released before the conference at the project web site:

`http://xprobe.sourceforge.net`

Acknowledgment

This study is conducted under the "III Innovative and Prospective Technologies Project" of the Institute for Information Industry which is subsidized by the Ministry of Economy Affairs of the Republic of China.

References

[1] O. Arkin and F. Yarochkin. A "Fuzzy" Approach to Remote Active Operating System Fingerprinting. available at http://www.sys-security.com/archive/papers/Xprobe2.pdf, 2002.

[2] D. Crowley. Advanced Application Level OS Fingerprinting: Practical Approaches and Examples. http://www.x10security.org/appOSfingerprint.txt, 2002.

[3] Fyodor. Remote OS detection via TCP/IP Stack Finger Printing. http://www.phrack.com/show.php?p=54&a=9, 1998.

[4] L. G. Greenwald and T. J. Thomas. Toward undetected operating system fingerprinting. In *WOOT '07: Proceedings of the first USENIX workshop on Offensive Technologies*, pages 1–10, Berkeley, CA, USA, 2007. USENIX Association.

[5] J. Jiao and W. Wu. A Method of Identify OS Based On TCP/IP Fingerptint. In *UCSNS International Journal of Computer Science and Network Security, Vol.6 No. 7B*, 2006.

[6] M. Kydyraliev. Openbsd ttl fingerprinting vulnerability. http://www.securityfocus.com/bid/4401, 2002.

[7] A. Luigi. Apache 2.0.39 directory traversal and patch disclosure bug. http://securityvulns.ru/docs3377.html, 2002.

[8] P. Ning, Y. Cui, D. S. Reeves, and D. Xu. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur.*, 7(2):274–318, 2004.

[9] G. Portokalidis and H. Bos. Sweetbait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Comput. Netw.*, 51(5):1256–1274, 2007.

[10] S. Shah. Httprint: http web server fingerprinting. http://net-square.com/httprint/httprint_paper.html, 2004.

[11] S. Shah. Ajax fingerprinting. http://www.net-security.org/dl/articles/Ajax_fingerprinting.pdf, 2007.

[12] F. Veysset, O. Courtay, and O. Heen. New Tool and Technique for Remote Operating System Fingerprinting. http://www.intranode.com/site/techno/techno_articles.htm, 2002.