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
Taipei, Taiwan
{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][13]. 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 fingerprinting test information gain weighting evaluation, which originally was 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 level. 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 low-level protocol packet variations[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]. The

[*]Yennun Huang is with Institute for Information Industry, Taipei, Taiwan

Xprobe2++ includes the analytical module that attempts to detect and identify possible Honeynet systems among the scanned hosts.

This paper primary contribution is introduction of remote network fingerprinting tool that uses both network level and application level 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.

Active Operating System fingerprinting is the process of actively identifying characteristics of the software (such as Operating System type, version, patch-level, installed software, and possibly - more detailed information), which runs on scanned computer system using information, which is leaked by the target system network stack.

The possibility of performing the active "fingerprinting" of network protocol stack exists because of the fact that actual protocol implementation within each operating system or software component may differ in details. That is because every operating system network stack is build in accordance to the protocol specification requirements (such as RFC), however every protocol may have some "gray" areas - states of the protocol, which are not covered, or not covered to the full extend by the protocol specification.

Additionally, there is a number of other critical factors that affect the efficiency and accuracy of the remote network mapping and active operating system fingerprinting scan. Some of these issues are relevant to the way the network mapping tools are designed, while other issues are more specific to the network topology of the scanning environment, underlying datalink type, type of network connectivity to the target and so on. Depending on the variation of the network configuration, the type of information that we would be able to collect regarding target system, would also variate.

In the remaining part of this section we are going to discuss typical problems and issues that a network mapping and active operating system fingerprinting tool has to deal with while performing the scanning process.

2.1. Handling Packet Filtering Nodes

When packets traverse across the network, there is a possibility that these packets (especially the malformed form of the packets, which are frequently used in OS fingerprinting signatures) would be modified, which affects the accuracy of the OS fingerprinting itself. Xprobe2++ tool is aware of this fact and fuzzy signature matching mechanism is designed to deal with this type of problems.

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.

2.2. Detecting Modified TCP/IP stacks

Some TCP/IP network stacks may be modified deliberately to confuse remote Operating System Fingerprinting attempts. 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. Xprobe2++ deals with the fact by using additional application level differentiative tests to map different classes of operating systems. The methods of application level fingerprinting are known to be effective[2] and when the application level tests are used along with network level tests, it is much harder to alter system applications to behave differently, because such behavior is dictated by the design of the OS underlying system calls or certain OS limitations. For example a test that uses 'directory separator' mapping simply tests how target system handles '/' and '\\' type of slashes to differentiate Windows hosts from Unix. Normalizing such behaviors at application level may lead to additional computational overhead, or may be not possible. For example normalization of "..\..\ request handling on web server running on the top of OS/2 platform may "unplug" a security hole on this operating system[7]. The Application Level Tests are further discussed in section 4.

2.3. Use of Malformed Packets

If a remote active operating system fingerprinting tool utilizes malformed packets to produce the fingerprinting results, there would be some sophisticated packet filtering devices that may drop these malformed packets or even block the scanning IP address. Therefore it is extremely important for a remote fingerprinting tool to provide the capability of performing remote fingerprinting using only valid network packets.

Malformed packets may also have another negative effect. Some malformed network packets might cause some TCP/IP stacks to crash or lead to excessive alerting by Intrusion Detection Systems.

One of the focuses of Xprobe2++ tool is to provide capability to use `normal`[1] network packets when possible to execute remote fingerprinting.

3. Tool Architecture Overview

The Xprobe2++ tool architecture includes several key components: core engine, signature matcher, and an extendable set 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.

Each of the modules is also responsible for declaring its signature keywords and parsing the data supplied in signature file.

The Xprobe2++ also expects modules to provide methods to generate signature entries in signature generation mode.

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

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[2].

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.

[1]We consider network packets that conform to the protocol specification to be normal.

[2]Please refer to the Xprobe2++ manual page and the Appendix Section for the detailed information.

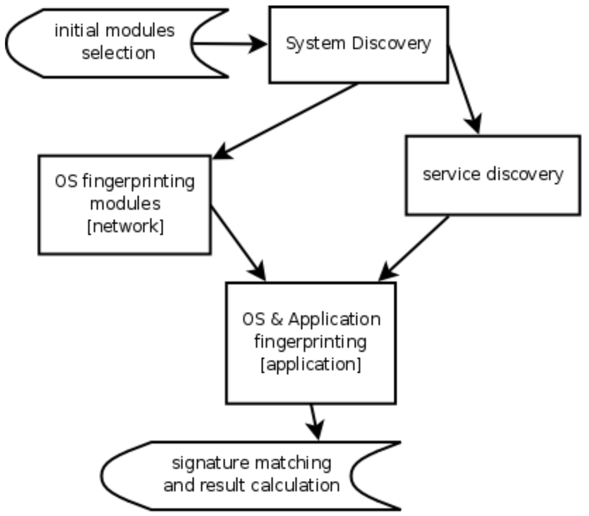
initial modules selection
System Discovery
OS fingerprinting modules [network]
service discovery
OS & Application fingerprinting [application]
signature matching and result calculation

Figure 1. Implementation Diagram

The aim of the 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. 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 provide set of tests for a target (with possible results, stored in Signature files), which primary purpose is 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 calculated automatically using the information discovered by network discovery modules.

3.3. Optional 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. Therefore the test execution and signature matching algorithms were designed not to execute network probes, which do not have sufficient information and simply relay on the results of remaining network tests. Xprobe2 and Xprobe2++ tools include a large number of tests that use different protocols(such as ICMP or SCTP), which do not require port numbers to execute.

Some other modules may require a valid port number, but make use of any response received from the target system, making the port scanning procedure redundant.

For example when SNMP query is sent to the target system and the target system UDP port is closed, the ICMP port unreachable response would be received. In this case the received datagram can be analyzed and matched with the set of "ICMP port unreachable" signatures. 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.

Still, The success of some of Xprobe2++ fingerprinting module tests may solely depend on knowing a valid open or closed TCP port number and a closed UDP port number. The application-level information collection modules may also require open/closed TCP or UDP port knowledge for proper execution. If such modules are intended to be executed, the port numbers have to be provided via command line switches, or the port scanner module has to be enabled and executed.

The port scanner module allows the tool to discover open and closed UDP and TCP ports, and later use the discovered port numbers as parameters for the operating system fingerprinting tests. The default behavior for Xprobe2++ is not to execute the port scanner module and the tool will not tie the port scanning module with the fingerprinting modules. This way Xprobe2++ still can maintain its minimal usage of packets for the network discovery.

3.4. RTT based Adaptive Mechanism

Xprobe2 uses adaptive mechanism that requires knowledge of Round Trip delay Time (RTT) - the elapsed time for transit of a network packet to the remote system and back. This delay time is measured when first responses are received from the target host. The RTT value is later accessed and can be used by the modules to identify requests that trigger no responses from target systems, or cases when responses are dropped or filtered by firewalling device.

The RTT number (in seconds) could also be specified from command line and this would override the measured value.

3.5. 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 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. Xprobe2 currently utilizes one of the simplest forms "fuzzy" matching derived from Optical Character Recognition (OCR) algorithms. With this algorithm a matrix of fingerprint matching results to signatures is filled in as tool progresses with probing and then final scores for each signature are calculated. The calculated scores and modules execution sequence is also affected by each module "information gain" weights. The higher the weight means the module result has greater impact on the final decision.

In Xprobe2++ each module weights are calculated using the number of available signatures using the algorithm proposed in [4].

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 level 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 "fuzzy" matching algorithm that matches signatures to received responses is based on a simple matrix representation, shown in figure 3.5, of the scan (or scans) results and OS signatures. The scan results and recorded for each test and the final score is calculated by a sum of 'matching scores' for each signature. The "matching score" is product of the test execution result, the module information gain weight and the reliability number. All tests are executed independently.

OS Test	OS 1	OS 2	...	OS i
Test 1	score(1)	score(2)	...	score(i)
Total	t(1)	t(2)	...	t(i)

Figure 2. Signature Matching Matrix

As the tool is progressing with target system probe, the matrix is being filled in with score values that signify how well the received response matches the signature of each operating system and adjusted by the module weight and reliability number in final decision making process.

When the scan is completed, the total score is calculated and the highest-matching signature (or a list of possibly matching signatures) is given as the final result.

4. Tool Improvements

The key Xprobe2++ improvement is inclusion of application-level modules, which allow tool to detect and correct possible mistakes of fingerprinting at network level. 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 level tests and network level tests may signify presence of a honeypot system, a network-level packet normalizer or a system running static port address translated (PAT) services.

The detailed list of implemented application level tests is shown in Table 4. As it can be observed from this table, some of these application level 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 given groups of application level tests, which are supported by our tool. However it should be understood that the testing possibility at application level is not limited by those methods discussed in this section. More specific application level 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.

- Directory separator tests, New line character handling tests, special file naming tests, special file names, case sensitivity testing, root directory names, special characters handling - 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 special way, file names are case insensitive, request to file names containing special character0x1a (EOF marker) will return different HTTP response from a web server running on the top of Windows (403) and Unix OS (404).

- File system limitation tests - This group of tests primarily aims at "identifying" particular underlying file systems using the knowledge of filesystem limitations specific for each filesystem. Such limitations may include maximum length of path, length of the file name and so on.

- 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. 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 time stamps returned by application and stored within the file and further identify the version of Microsoft Windows Operating System, because different versions of Windows format the file differently. Existence of ".file" along with actual "file" reveals MacOS operating system.

- Binary file fingerprinting - If it is possible to obtain binary files (for example from anonymous FTP), it would be possible to predict the type of host operating system by analyzing the binary format of downloadable binary files. This method is not currently implemented with Xprobe2++ as it may trigger bulk download of massive amounts of data.

- Specific/Default Applications - It may also possible to identify target operating systems by mapping subsets of default applications to operating system stacks. This method is still under research and is not implemented in Xprobe2++.

  To implement this method reliably, the Application software methods may also have to be reliably implemented. The version detection based on software banners (when available) is extremely unreliable, however other techniques, such as those discussed in [10] could be used to fingerprint application versions reliably.

- Information Leaks as additional Source of information - Aside from the actual OS identification, it is often

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
Specific/Default Application Software	many protocols	Windows types, Unix types, Other

Figure 3. Xprobe2++ Application Level Tests

possible to collect additional data from the target system by using variety of tests. For example, there are information leak bugs in some network stacks, when the stack leaks part of the system memory data as "padded" data. These methods are currently being researched as well.

At this moment Xprobe2++ includes some of modules that collect such information, however currently they do not have any signatures and do not contribute to the process of actual operating system fingerprinting.

We also believe it might be possible to perform further differentiation of operating systems based on encoding types, supported by application or underlying file system. It may also be possible to analyze distribution of application level response delays for different requests in order to identify "fake" services or fingerprint particular software versions. Further research in this area is needed and at the present moment we have not been able to design reliable methods of fingerprinting using these types of tests.

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, cisco 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.

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 4. Please note that nmap needs to execute port scanning in order to be able to successfully guess remote operating system type, while Xprobe2++ can relay on results of the tests, which do not require any ports to be known.

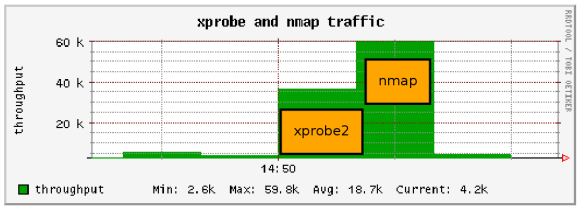

Figure 4. Xprobe2++ and nmap generated traffic loads.

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-level tests is that the tool 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-level tests on filtered port numbers. We believe that preliminary port probe for each application-level test may be helpful to resolve this limitation.

Further, since the tool is issuing potentially malformed application level requests, this may open additional possibilities for tool activity detection and possible crashes of probed application. We should note that we did not observe any abnormal behavior on tested applications, however this does not guarantee that no application would crash after receiving malformed protocol requests.

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, since there are some issues with libpcap reenterancy and performance. 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

Although some advancement was made in the field of operating system fingerprinting in the past few years, there is still plenty of space for improvement of these techniques.

We hope that by releasing new generation of the tool we will take active operating system fingerprinting to the next level of its evolution.

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.

[13] J. Zhou, M. Heckman, B. Reynolds, A. Carlson, and M. Bishop. Modeling network intrusion detection alerts for correlation. *ACM Trans. Inf. Syst. Secur.*, 10(1):4, 2007.