Detecting and Defending against Web-Server Fingerprinting

Dustin Lee, Jeff Rowe, Calvin Ko, Karl Levitt

Computer Security Laboratory,
University of California, Davis

NAI Labs,
Network Associates, Inc.

{leed,rowe,levitt}@cs.ucdavis.edu, calvin_ko@nai.com

Abstract

Cyber attacks continue to increase in sophistication. Advanced attackers often gather information about a target system before launching a precise attack to exploit a discovered vulnerability. This paper discusses techniques for remote identification of web servers and suggests possible defenses to the probing activity. General concepts of fingerprinting and their application to the identification of Web servers, even where server information has been omitted are described and methodologies for detecting and limiting such activity are discussed.

1. Introduction

> "Note: Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementors are encouraged to make this field a configurable option." [RFC 2068: HTTP/1.1 (section 14.38)]

Knowledge of software vendor and version information provides a huge advantage to attackers in penetrating a system. In a sophisticated cyber attack, information about potential vulnerabilities in the target system are first gathered then a penetration attempt is launched. Although the security of a system should not be based solely on obscurity, it is desirable to protect against attempts to gain system information.

HTTP servers and clients[1] exchange meta-data during a transaction in the form of MIME-like headers. For instance, within a Request a client can inform the server what level of HTTP it is using and what format of return data it prefers and is willing to accept. A server typically responds with headers describing, among other things, the size, format, name and transfer time of the returned document. It is important to note that this meta-data is not authenticated in anyway. If a server sends incorrect format information the client will likely misinterpret the message. Clients and servers are designed to cooperate and avoid this situation.

Some information normally sent within the HTTP headers is security sensitive. If an attacker has information about the vendor and version number of a server (not to mention server plugins) this information could be used to exploit vulnerabilities more efficiently. Unfortunately, few vendors have taken the advice offered in the quote above.

Since HTTP servers do not generally provide a simple means for modifying or omitting the server's self-description (e.g. as a configuration file option) a server administrator can not easily manipulate this information. However, through modification of source code (or even the binary using reverse engineering) it is possible to remove or obscure the server header from an HTTP response. A small percentage of servers on the Internet currently do just this. This paper will demonstrate that even after hiding the server information a determined attacker can still learn the server's vendor and version number with a high degree of confidence. The HMAP method uses fingerprinting techniques to gather information about a server's identity from HTTP headers returned from a variety of "normal" and "abnormal" Requests.

This paper describes techniques for fingerprinting HTTP servers and discusses possible defenses to the activity. HTTP servers have been a popular target of penetration, largely because they are one of the most common web services to be running and publicly available. In addition,

[1] also known as web servers and browsers

many vulnerabilities exist in HTTP servers[6]. As with most types of software, the vulnerabilities in HTTP servers are specific to vendor and version. Previously, researchers have investigated fingerprinting on TCP/IP and defenses to the activity [14][15]. Nevertheless, this paper focuses on the application layer and the HTTP protocol.

In viewing the fingerprinting problem of networking software, we identify several general concepts: lexical, syntactic, and semantic analysis of the Response and discuss their effectiveness in identifying the identity of the software. We have built a tool (HMAP) for testing existing HTTP servers and discovered that most of them can be precisely identified. In addition, we discuss approaches to the problem of the availability of HTTP based fingerprinting tools.

The organization of this paper is as follows: Section 2 provides an overview of existing fingerprinting tools. In Section 3, we describe several methods for identifying web servers. Section 4 suggests possible approaches to remedy the problem. Finally, the security implications of this work will be discussed.

2. Related Work

This section reviews existing fingerprinting and web server investigation tools. These applications work by using HTTP and other internet protocols in non-standard ways.

2.1. Web Server Surveys

Netcraft [7] and SecuritySpace [8] are two groups that perform periodic surveys of the web to determine the market share of various web servers. One feature available from their sites allows a user to determine what HTTP server and OS a particular site is running. After requesting this service for a workstation who's TCP/IP traffic was being monitored by the author the Request in figure 1 (with modified Host line) was observed.

Since most servers' responses have a "Server" header including the vendor information it is a simple matter to extract this information. Presumably this same method is used to perform their web-wide survey. As already discussed, server information determined this way is not very reliable. In fact, some simple changes to Apache/1.3.12 (Win32) source code fooled their HTTP server identity checker. Interestingly, Netcraft is able to deduce the OS as well. This information is most likely acquired from the TCP/IP level using techniques similar to that of NMAP.

2.2. NMAP

NMAP [3] is a well-known network probing tool. In its default mode it performs a simple port scan to determine which ports are active on a target system. With the -o option it attempts to detect the OS and its version number. It does this by using known bugs and vagaries of different vendors' implementations of the TCP/IP stack. Most OSes can be uniquely characterized by comparing the target's behavior with known profiles.

For instance, the correct behavior on receiving an unexpected FIN flag is to ignore it and not respond. Several operating systems incorrectly reply with a RESET. Using a decision tree method the specific operating systems can be identified by correlating the responses received after a series of similarly constructed messages. Depending on the behavior observed (like the FIN probe just described) movement through the tree continues until a leaf is reached. In most cases this answer is unique.

The results obtained by this method are not conclusive. It is entirely possible that a TCP/IP stack could be specifically rewritten that spoofs the behavior of any given OS or of none at all. Going to this extreme is fairly unlikely since it is always risky to alter software that is working correctly (or correctly enough).

2.3. PRO-COW

Krishnamurthy, et al use an unreleased tool [2] to monitor web servers for compliance with HTTP/1.1 specifications. While this tool was not designed for security or identity probing purposes, some of its functionality is similar to that of HMAP. For instance, the PRO-COW tool spoofs various activities of a client as part of its series of tests. These tests check both for appropriate default behaviors as well as appropriate responses to error conditions. As an example, HTTP/1.1 Requests MUST [1] include a "Host" header identifying the domain to which the URI pertains. If a client doesn't include this field a "400 Bad Request" error should be returned.

As the PRO-COW project has demonstrated many servers are at different levels of compliance. Since these differences tend to be constant for a given vendor/version number combination this information can be used for obtaining clues about server identity, though they do not report using it in this manner.

2.4. whisker

whisker [4] uses unusual HTTP Requests to probe web servers for known CGI based vulnerabilities. By sending specially crafted HTTP Requests, whisker analyzes the Responses to check for signs of vulnerable software. The Request variations employed are primarily within the Request line URL. Of specific relevance to HMAP, this tool has been tuned to elude typical IDS monitoring procedures. For instance, requested URIs can be URL-encoded so that simple string matching techniques will fail. While whisker

```
HEAD / HTTP/1.1\r\n
Connection: close\r\n
Referer: http://www.netcraft.com/survey/\r\n\r\n
User-agent: Mozilla/4.0 (compatible; Netcraft WebServer Survey)\r\n
Host:  local.host.edu\r\n\r\n
```

Figure 1. Netcraft Surveys Request

uses information about the specific OS and web server being probed to tailor its vulnerability probe, it relies on the server's self-description of its identity in the "Server" field. Whisker might be even more effective if it had a method of verifying the vendor and version number of the host it is communicating with.

3. Concepts and Methodology

Fingerprinting is a heuristic method of observing the behavior of a software component for the purpose of determining its identity. In general, fingerprinting involves sending specific requests to the component and observing the response.

For HTTP servers the content of Responses to a range of specifically chosen Requests are sufficient to determine their identity with high confidence. As a proof of concept, a tool called HMAP was developed to automate this process. The name HMAP was, of course, inspired by NMAP. However, whereas NMAP manipulates and analyzes the TCP/IP layer to gather clues about an operating system, HMAP performs at the HTTP level.

A typical method used to acquire a server's identity is illustrated in figure 2. This command simply pipes the string "GET / HTTP/1.0\n\n" to the netcat command which opens a TCP/IP connection on the default HTTP server port of the target server. The reply from the server is filtered by the grep command and only lines containing the text "Server" are displayed. In most cases this will yield something like the following:

```
Server: Apache/1.3.12 (Win32)
```

Generally, this method yields accurate information, but there is no guarantee. In fact, it is a simple matter to cobble together another "netcat" script on the server end that responds with an arbitrary "Server" header.

A natural first attempt at hiding server identity is to simply not send it in the first place. No server, to our knowledge, has this as a configurable feature[2] but with access to source code it is easy to change, either by clearing it out or setting it to a fictitious value. Using a variation of the above "netcat" technique an informal survey of the top 100 most visited web sites [12] was conducted. The results indicate that about 5% of sites[3] use this sort of hiding method (i.e. not sending back server headers or leaving them blank). This tactic will only work against an uninformed attacker, because methodically fingerprinting a server, its true identity can be determined with a high degree of confidence.

3.1. Methodology

To generate a server's fingerprint a set of characteristics that discriminate a specific server's use of HTTP from that of other servers must be identified. These differences will arise from variations in how closely a vendor has followed the HTTP specifications. The specification for the different HTTP versions (e.g. 1.0, 1.1) and features (e.g. cookies, DAV, S-HTTP) are found in RFCs[4]. These specifications are not enforced, rather they are agreed upon conventions. If a vendor does not comply with the specifications it risks incompatibility with the software of other vendors. Of course, if others deviate from the specification in the same way then this deviation itself becomes a *de facto* specification (e.g. cookies developed in this way with Netscape taking the lead).

The HTTP RFCs describe the features of the HTTP protocol in terms of varying levels of compliance. The words MUST, SHOULD and MAY (and each of these in combination with NOT) are used to indicate the importance the RFC's authors placed on individual features. Implementations that are deficient in the MUST category are considered to be non-compliant. A given web server may function acceptably and still be non-compliant. It is, in part, this variability of compliance, that allows HTTP servers to be fingerprinted.

Employing the concept of compliance variations, a list of characteristics to use for fingerprinting can be assembled. For each of these characteristics a test (i.e. HTTP Request) must be designed that will provoke a Response exhibiting the characteristic. The list of characteristics can be derived by examining how a server responds under various condi-

[2] with the partial exception of Apache which at least allows the amount of server information to be reduced

[3] This only includes sites that are leaving the Server header blank or just not sending it at all. Others could be lying about who they are but HMAP has not been used against the others for verification

[4] Request For Comment

```
echo -e "GET / HTTP/1.0\n\n" | nc www.someserver.com 80 | grep Server
```

Figure 2. netcat "one-liner" for identifying a server

tions. The list of possible identity discriminators discussed below is not exhaustive of all but it is representative of the most common varieties. Other variations may also be developed that contribute to the complete fingerprinting picture.

3.2. Fingerprint Characteristics

Using the analogy of programming language classifications, the types of characteristics that an HTTP fingerprinting methodology incorporates can be divided into the following categories: lexical, syntactic and semantic.

- Lexical: specific words, phrases and punctuation that are used in Responses.
- Syntactic: ordering and context of elements (e.g. words, phrases, headers).
- Semantic: a server's specific interpretation of a Request from among the possible interpretations.

Sufficient information for discrimination can be gathered by focusing on a subset of characteristics and test variations. Note that it could be argued that some of the characteristics can not be uniquely classified (is "\n" a lexical element or syntactic?). Some vagueness can be tolerated as the classification's primary aim is simply to guide and organize the search for identifying characteristics.

3.2.1 Lexical

The lexical characteristics category covers variations in the actual words/phrases used, capitalizations and punctuation. These differences tend to be conspicuous and are quite useful for fingerprinting.

Response Code Message An HTTP Response includes a numeric value describing the success or failure of the server's attempt to satisfy the Request. For each of these error codes there is corresponding human readable text. For instance, for the error code 404, Apache reports "Not Found" whereas Microsoft IIS/5.0 reports "Object Not Found". While some messages are fairly uniform across the various implementations (e.g. 200 rarely deviates from "OK"), there is generally a fair amount of variation between different servers, which is manifest both in the actual words used and in the capitalization pattern (all initial capitals versus only first word capitalized is a common distinction, e.g. "Not Found" vs. "Not found").

Header Wording Generally header names must exactly match those in the specification or the client will be unable to identify them and correctly interpret the Response. Regardless of this constraint, variation still occurs in the form of capitalization patterns. For instance, the header "Content-Length" is returned by some servers and "Content-length" is returned by others.

Line Terminators Some servers use only "\n" to separate elements of the header while the RFC [1] specified behavior is to use "\r\n". This sort of deviation is less common in servers under current development but can be found in some earlier offerings.

Server's Name While the purpose of HMAP is to be suspicious of the server's reported identity, the server's claim of its identity can still be used as one of the characteristics.

3.2.2 Syntactic

HTTP messages are required to have a predefined structure so servers and clients can understand each other. Nonetheless, some variation in the ordering and format of Request elements like the headers and their contents is found.

Header Ordering The HTTP specifications suggest that HTTP Response header fields should be ordered as followed: "general" followed by "response" followed by "entity". Different vendors follow this suggestion to various degrees but even those that do still differ in the ordering within these categories. For instance, Apache servers consistently place the "Date" header before the "Server" header while Netscape-FastTrack/4.1 have these headers in the reverse order. The ordering of the entire set of headers sent by a server provides useful fingerprinting data.

List Ordering There are many instances where the contents of a header will be a list of items. For instance, when an OPTIONS method is sent in an HTTP Request, a list of allowed methods for the given URI are returned in an "Allow" header. The order of these elements tend to vary between servers. This is also true for the "Vary" header. Not all lists are useful for discriminating identity, however. For instance, the Content-Language header can identify more than one human language type. The ordering may imply the order of percentage of each language. In this case the ordering is descriptive of the returned content and not an artifact of the inner-workings of the server.

Formatting Some elements of the headers have formats that are variable or unspecified by the RFCs. For instance, the "ETag" header provides a unique identifier (such as a hash) for a given document that can be used, among other things, to determine if the client has already seen this document. For instance Apache/1.3.11 returns an ETag header with the following format: "0-574-38379154;3a5b7811". The Jigsaw/2.1.2[10] server returns ETags such as the following: "mvanct:s0jndthg". Since there are no official guidelines for how ETags should be constructed and presented these tend to be a good characteristic for fingerprinting.

3.2.3 Semantic

When a server receives a Request it must first decide on an interpretation for the Request before it attempts to satisfy it. In addition, when the server constructs the Response and assigns the return code it has to assess if the request was satisfied properly and if not, what was the cause of failure. Furthermore, the server has to decide what information to send back to the requester in the form of response line, headers and body. There is tremendous variation on how servers interpret both well-formed and mal-formed Requests.

Existence of Response Line and Headers Some requests will cause the server to believe the requester is an HTTP/0.9 based client. Legal responses to HTTP/0.9 clients are only allowed to include a body (i.e. no headers or Response line). The existence or non-existence of a header can be used to infer how a server interpreted a client Request.

Presence of Specific Headers A server has a choice of headers to include in a Response. While some headers are required by the specification, most headers (e.g. ETag) are optional. For instance, upon a "501 Method Not Implemented" error Apache servers send an "Allow" header with a list of the allowed methods for the designated URI, while Jigsaw/2.1.2 does not.

Response Codes for *Ad Hoc* Requests Even when most servers agree that a certain Request is malformed, they often assign it a different type of error. For instance sending the text stream "hi" (with no headers or other HTTP trappings) to an Apache server provokes a headerless response whose message body warns that the method "hi" is not implemented. Microsoft IIS/5.0 replies with "400 Bad Request". This implies that Apache interprets the Request as a bad reply from a HTTP/0.9 client whereas Microsoft takes it as a malformed Request from an HTTP/1.X client.

There are a large number of possible variations for developing *ad hoc* test requests. The main elements that can be manipulated are the method line, the headers and the body (e.g. size and whether or not it is included). For instance, the method line normally contains a method, a URI and a version. Each of these is separated by "whitespace" with nothing preceding the method and a line terminator immediately following the version. Creating malformed variations is a simple matter of employing misspellings, switching positions, omitting elements, varying the quantity and type of "whitespace" and so on. Figure 3 shows a partial list of some of the ways that the method line alone can be varied.

Different servers have very distinctive reactions to these Requests. A unique fingerprint can be developed simply by issuing these "naked" method lines one at a time against a target server.

Error Ranges For Requests and Request objects (e.g. headers) of unusually large length (unusual being defined by the server) the HTTP/1.1 specification provides the errors "413 Request Entity Too Large" and "414 Request-URI Too Long" respectively. If a binary search is performed to identify the length of the URL that initially provokes each type of error for a given server, it becomes clear that different servers have unique patterns of errors that they pass through for increasing sizes. Table 1 shows the lengths of URLs that elicit different errors for two servers.

Note that not only do these two servers report that length limits have been passed for different URL lengths but they also report them as different problems. Apache indicates that the URI itself is too large whereas Netscape reports that the entire request (including headers and body) is too large. There are many tests that provide a similar behavior for monotonically increasing parameters. For instance, the number of headers or the length of individual headers can be varied to achieve similar identifying patterns.

3.3. Tool Usage

HMAP is a program that automatically performs a wide variety of tests similar to those described above. It analyzes all of its interactions with the target server and compares each of the responses with a list of known server characteristics. The comparison method is simple but effective. For each test it attempts to provoke a response from the target. If it fails to receive a response (for instance some errors aren't even implemented in some servers) then it notes which characteristics it wasn't able to compare. If it does get a response then it determines whether or not the match was exact and scores accordingly. For each server that HMAP has a profile, it displays a final result of how many exact matches, misses and "don't knows" were found. For instance, one line of the output might be:

```
Apache/1.3.11 (Win)        21:9:5
```

```
"GET"                    "GET / HTTP/Q.Q"                  "HEAD    /       HTTP/1.0"
"GET /"                  "GET / HTTP/"                     "HEAD //////////// HTTP/1.0"
"GET / HTTP/999.99"      "GET/HTTP/1.0"                    "Head / HTTP/1.0"
"GET / hhtp/999.99"      "HEAD /.\ HTTP/1.0"
"GET / http/999.99"      "HEAD /asdfasdf/../ HTTP/1.0"
"GET / HTTP/Q.9"         "HEAD /asdfasdf/.. HTTP/1.0"
"GET / HTTP/9.Q"         "HEAD /././/././/././/./ HTTP/1.0"
```

Figure 3. *Ad Hoc* **Request Examples**

Server	URL Length	Response
Apache/1.3.12 (Win)	1-216	404 Not Found
	217-8176	403 Forbidden
	8177-up	414 Request-URI Too Large
Netscape-FastTrack/4.1	1-4089	404 Not found
	4090-8123	500 Server Error
	8124-8176	413 Request Entity Too Large
	8177-up	400 Bad request

Table 1. Long URL Ranges

where the score in this case indicates that 21 characteristics were an exact match for Apache/1.3.11 (Win), 9 were definitely different and 5 couldn't be determined. One score line appears for each server that has been previously profiled. Whichever server (or set of servers) matched most closely by having the highest number of exact matches will be noted as such. If one attempts to fingerprint a server for which there is no pre-existing profile then a good match shouldn't be expected (unless the server happens to behave like another known server). The result of this process is that a closest matching server type for a target can be determined. This is not proof of identity but is a good indicator of such.

It is important to note that it is not possible to discriminate between two different server instances of the same type. For instance, two Apache 1.3.12 servers that are configured exactly the same will appear as identical targets to HMAP.

3.4. Assumptions/Limitations

HMAP makes the reasonable assumption that the target server behaves deterministically. Perhaps this seems like an obvious assumption, but one technique for frustrating identity probing is to randomize responses to some extent to make it more difficult to compose a reliable fingerprint of the server. In general, this assumption has been reliable for all of the tested servers. HMAP also assumes that that the features it tests for are not easily configurable to other settings or if they are, it is rarely done. This assumption affects which tests can be reliably used. In practice only tests that produced consistent fingerprinting discriminators were retained in the testing set.

HMAP performs all available tests even if all tests completed so far match only one server. In theory a decision tree technique might seem more efficient. This method has been avoided since the server's identity is not known *a priori* and potential matches shouldn't be disregarded. In addition, one change to a server could easily frustrate a decision tree method. By testing all characteristics a more accurate picture develops.

4. Security Implications

Web servers are vulnerable to a wide variety of attacks many of which are targeted by vendor and version number. A major contribution of the HMAP methodology is to underscore the difficulty that should be expected in thoroughly hiding a server's identity in an attempt to avoid such attacks. In this section the security implications of the availability of HMAP-like tools will be discussed. At first glance HMAP may seem to be of primary benefit to attackers. However, attackers do not currently need this sort of tool because very few sites hide their identity. In addition, it is frequently the case that attackers have more information (and more current information) on vulnerabilities than the defenders, so making the information available is more likely to help than hurt.

4.1. Attackers

It might be argued that the availability of a tool like HMAP does not impact web server security since even if a site is actively trying to hide it's identity an attacker can simply try all known attacks and observe which, if any, succeed. However, without a method similar to HMAP the attacker is forced to increase both time and bandwidth used for an attack which generates more data for an Intrusion Detection System to use. Using an HMAP like method for gaining information about a system allows an attacker to narrow his attack space. While probing for system identity will generate traffic, probing behaviors are less likely to generate alarms and have a more ambiguous legal status than an actual attack. The actual attack can then be launched at another time and from another location.

Running the full suite of HTTP server fingerprinting activities can be fairly easy to detect, thus, it should be expected that an attacker concerned with stealth would attempt to decrease the fingerprinting visibility. The following are examples of techniques to accomplish this:

- running subsets of tests from several computers and correlating data. IDSes might not detect a pattern of behavior if it analyzes records on a host by host basis.
- running tests over a long period of time. If the IDS notices one unusual Request then it might attribute it to random error and the observation will be flushed after some time.
- doing only short forms of large Request tests (e.g. first half of long URL test list may provide enough information for discrimination)
- making a search tree of characteristics and only doing the necessary ones. A minimal subset of characteristics could be assembled that identified a specific server more efficiently.
- masking contents of Requests with URL encoding. Some IDSes do not perform URL decoding.
- changing the contents of long URL type Requests so they vary in more than just the length. Currently the long requests are composed of all the same characters and do not look like legal Requests.

A further enhancement to an HMAP-like tool would be for it to automatically attempt exploits once the server type has been determined.

Ascertaining a server's identity this way is not necessarily against the law or even a precursor to an attack. Even if a system administrator were to detect that someone was running HMAP against their site it is not clear what recourse they could take[16]. Information gathering and probing is currently a gray area within the security community (and legally within different countries) and likely to remain so for some time.

4.2. Defenders

The different aspects of computer security generally fit into one of the following categories: protection, detection and reaction. System Administrators and HTTP server developers frequently work at the protection level, while Intrusion Detection falls in the detection level. Reaction is beyond the scope of this work. Techniques for improving system security using insights gained from the HTTP fingerprinting methodology are now discussed. The most successful protection will include elements from each of these complementary categories.

4.2.1 System Administrators

There are already many resources related to security configurations for web servers (e.g. [11]). Instead of a general discussion of web server security techniques, this section will focus on the more specific case of security issues that arise from the availability of HTTP server fingerprinting tools.

The most obvious use a system administrator could have for the HMAP tool would be to use it against his own servers to determine if they can be readily externally identified. Unfortunately the ability to hide a server's identity will depend greatly on access to the server source code and what configuration options were made available by the vendor.

System administrators should also be aware of log related issues. While not unique to the HMAP tool, it is very easy to fill up log files at an abnormal rate with HMAP type Requests. If there is no mechanism in place to control the lengths of the logs, then testing with long URLs can quickly consume disk space which might be part of a denial of service attack. Even if there is no danger of exhausting disk space, forcing large amounts of data through log files can obscure other attacks that might have occurred. If the log files do roll over automatically then forcing large amounts of data through could flush out signs of previous attacks.

Finally, since HMAP performs a large number of tests using non-standard Requests, it is useful for running checks for susceptibility to some forms of denial of service attacks. At least one server was identified that crashes consistently when HMAP was run against it.

4.2.2 HTTP Server Developers

HTTP server developers can assist with identity hiding by providing options to make server fingerprinting more difficult. For instance, configuration parameters or even compilation settings for source code could be made available

that allowed system administrators to control the amount and type of identity information that was returned in a Response. Developers can use details of the server fingerprinting methodology to identify the types of characteristics and behaviors that can be adjusted in order to hide a server's identity. The following are some strategies for effective identity hiding.

Common Interface and Behavior Fingerprinting succeeds because each vendor has a slightly different interpretation of how an HTTP server should appear and react. If the server development community agreed on a common interface, then fingerprinting would be a more difficult process. The HTTP specification, in a sense, is a step in this direction but to be effective different fingerprinting cases would have to be specifically addressed to guarantee uniformity. This is unlikely considering the greatly varying compliance with the much less stringent HTTP/1.X standards. As web server attacks increase in severity and frequency this sort of strategy may become more attractive.

Configuration Options It is uncommon to find any server configuration options that allow identity hiding. Apache approaches this capability by providing the server directive "ServerTokens" that allows for the truncation of the Server header's level of detail, but not for hiding it entirely. It would be beneficial if users could remove this line entirely. This is not a fool-proof identity concealment method but it is a step in the right direction. The ability to address the various characteristics used in fingerprinting would be more beneficial. By using configuration or build options the user should be able to have more control over the lexical, syntactic and semantic elements of Responses. These configuration options could allow users to masquerade as another server or act like no known server. In addition they could choose to match a generic standard as mentioned above.

Variable Output Another way to make fingerprinting more difficult is to use variable responses. For instance, "File Not Found", "Not Found", "Not found" could all be used randomly in appropriate Responses. Fingerprinting becomes more difficult if a static picture of the target can not be developed. This should not affect the ability of a client and server to converse since each of the variations would be semantically equivalent.

4.2.3 Intrusion Detectors - Misuse Detection

Like server developers, IDS developers can use details of the fingerprinting methodology to identify characteristics and behavior that indicate identity probing is occurring. To our knowledge there are currently no IDSes that look for web server fingerprinting activity. IDSes that look for CGI attack probes do exist and presumably could be extended to include awareness of HTTP-based identity probing.

As with many probing activities (e.g. port scans) it is not necessarily the case that HMAP type behavior is a guarantee that illicit activity will follow, but it should raise suspicions. Therefore an IDS would not want to automatically issue an alert upon every detection of this behavior. It will be the responsibility of a more comprehensive system to determine what to do with this information (e.g. decide that a probe might be part of a larger picture indicating attack preparations).

Two information sources which an IDS's sensors can monitor are TCP traffic and logfiles. Evidence that fingerprinting is occurring can be found in both these sources in the traces left by Requests and Responses. The suggestions below are geared towards misuse detection techniques, but anomaly and specification detection are also possible. Many of the techniques used by HMAP are not very subtle and would be easy to detect, but only if system administrators are aware of the goals of this type of unusual activity.

HTTP Requests

Request Element Size Many of the tests used to provoke server responses use very large elements. For instance large URIs and large numbers of headers are used to determine the Request sizes at which a server starts reporting certain errors. An IDS should look for these sorts of abberations especially when the message changes in size over a wide range and contains headers/URLs that are not typical.

Unknown and Unusual Elements Unknown methods (e.g. "QWERTY") or methods that normal browsers rarely or never send (e.g. "TRACE") should be detected. The same observation applies to unknown or unusual header fields.

Unusual Constructions Most Requests have a fairly simple format (e.g. method line of METHOD URI HTTP-VERSION followed by common headers). Unusual constructions, such as a Request including an inappropriate body or the use of incorrect line terminators should be examined.

Method Line Syntax Most browsers are fairly well behaved regarding how they issue a Request. Unusual spacing or corrupted version information is highly suspect.

Browser Behavior In the same way that a client can fingerprint a server, a server can check the structure and content of a client's Requests and determine if the client

is correctly specifying it's own identity. If a client's User-Agent field indicates one type of browser but it behaves like another, this should raise suspicion. On the other hand, some privacy "sanitizing" packages [13] purposely hide the name of the browser so this method might raise too many false alarms. At this time few users employ this sort of obfuscation so it should be a useful technique.

General IDS Eluding Techniques There are a number of standard techniques that CGI vulnerability scanners use to thwart IDS detection. An important example of these is URL encoding which allows for a Request URL to be rewritten using a hexadecimal format making pattern matching more difficult and CPU intensive. An extensive list of these sorts of techniques can be found at the whisker site [5]. Other IDS eluding techniques include sending Requests from multiple servers and correlating the results later. Since IDS systems do not have infinite capacity for keeping state, allowing long time spans to pass between Requests can also be effective.

HTTP Responses

Unusual and Repeated Errors Many of the fingerprinting tests are attempts to provoke non-"200 OK" responses. While some of these errors like "404 File Not Found" are fairly common, others like "413 Request Entity Too Large" are rare enough to raise suspicion. Even common errors like seen far out of proportion to their norm should also raise some flags.

Headerless Responses Since HTTP/0.9 type clients are fairly rare these days, if a server sends back a response that doesn't have a header it is possible that someone is masquerading as such a client or a request that has confused a server was sent.

Miscellaneous Techniques Several other techniques can be used together with traditional IDS sensors to augment system security. An IDS could also help prevent the fingerprinting by detecting "bad" Requests (too long, malformed etc.) and converting them before they are received by the server so that the interrogator can not correlate Requests with Responses correctly. This conversion could be done by the IDS system itself or by some sort of proxy. Similar techniques for preventing TCP/IP fingerprinting are described in [14] and [15]

Honeypots are another useful resource for tracking attacker behavior. With respect to web server fingerprinting a honeypot could detect a fingerprinting sort of behavior and then return purposely misleading responses. Later if an attack is launched against the type of server that was falsely advertised, the deception could be continued, perhaps letting the attacker believe that their attack was successful. This method would likely only be employed in situations where there is a strong need to learn about an attacker.

As with all intrusion detection techniques additional resources (time, code complexity, memory) will be required to monitor, translate and analyze HTTP traffic. It is important to balance the cost of the security with the actual value of the resources being protected.

4.3. Miscellaneous Use

As discussed earlier, Netcraft (as an example) performs Internet wide surveys to determine statistics on the usage of various servers [7]. With respect to server identity they use the simple HEAD method and check the "Server" line of the returned header. Most likely their current results are fairly accurate. If for security reasons the "Server" field becomes hidden more frequently then fingerprinting methods could allow them to continue to discover or verify their results.

Another more esoteric use relates to determining a "family tree" of sorts between servers. Some commercial servers are descendants of other server implementations. Suppose company XYZ based its server on a version of Apache. Depending on the changes, it might be possible to determine from which version of the server it branched. This in turn might point to existing security problems in that product if they are also known to exist in its parent.

5. Conclusion

Hiding the identity of an HTTP server from a sophisticated and knowledgeable attacker is a non-trivial endeavor. The content and organization of strategically provoked Responses carry enough information, in most cases, to uniquely fingerprint a web server's vendor and version information. Tools, such as HMAP, can be developed that mimic the behavior of a client to elicit such Responses and then automatically analyze them to create a fingerprint and identify a server. This can be a strong aid to an attacker since knowledge of the server's identity helps determine possible vulnerabilities to exploit. Alternatively, it is possible to strategically augment server security using the details of how such a tool would work. In either case it is important that security workers be aware of the possibility of this sort of information probing.

References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2068

[2] Balachander Krishnamurthy, Martin Arlitt. *PRO-COW: Protocol Compliance on the Web—A Longitudinal Study*, 2001

[3] fyodor. *Remote OS detection via TCP/IP Stack FingerPrinting*, http://www.insecure.org/nmap/nmap-fingerprinting-article.html

[4] Rain Forest Puppy. *whisker*, http://www.wiretrip.net/rfp/bins/whisker/

[5] Rain Forest Puppy. *A look at whisker's anti-IDS tactics*, http://www.wiretrip.net/rfp /pages/whitepapers/whiskerids.html

[6] Various. *http://www.securityfocus.com*, web

[7] Netcraft Surveys. *http://www.netcraft.com*, web

[8] Security Space. *http://www.securityspace.com /s_survey/data/index.html*, web

[9] Apache Group *http://www.apache.org*, web

[10] Jigsaw W3C's Server *http://www.w3.org /Jigsaw/*, web

[11] Security Tips for Server Configuration *http://httpd.apache.org/docs/misc /security_tips.html*, web

[12] 100hot Web Rankings *http://www.100hot.com /directory/100hot/*, web

[13] Internet Junkbuster Proxy *http://www.junkbusters.com/ijb.html*, web

[14] M. Smart, G. Malan, F. Jahanian. *Defeating TCP/IP Stack Fingerprinting*, 9th USENIX Security Symposium

[15] D. Watson, M. Smart, G. Malan, F. Jahanian. *Protocol Scrubbing: Network Security through Transparent Flow Modification*, DISCEX '01. Proceedings, Volume: 2, 2001

[16] Can I take legal actions against port scanning? *http://www.sans.org/newlook/resources/IDFAQ /port_scanning_legal.htm*