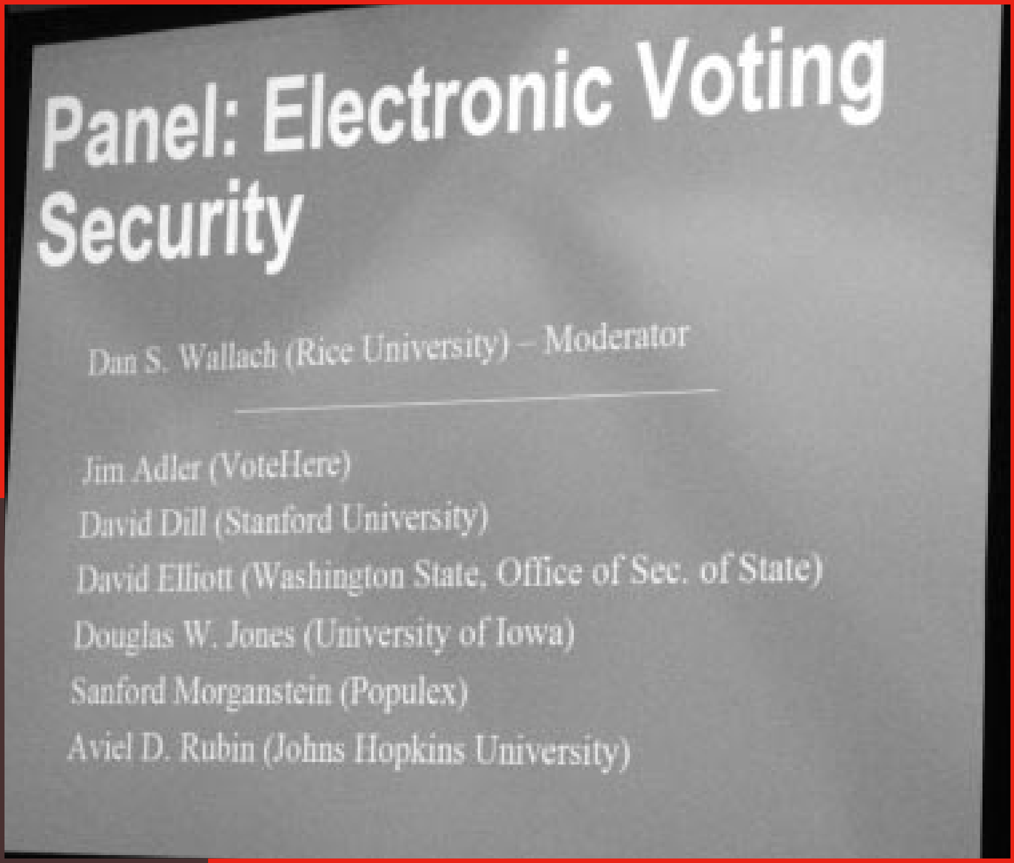
;login:

THE USENIX MAGAZINE

December 2003 • volume 28 • number 6



Panel: Electronic Voting Security

Dan S. Wallach (Rice University) – Moderator

Jim Adler (VoteHere)
David Dill (Stanford University)
David Elliott (Washington State, Office of Sec. of State)
Douglas W. Jones (University of Iowa)
Sanford Morganstein (Populex)
Aviel D. Rubin (Johns Hopkins University)

inside:

Focus Issue: Security

Guest Editor: Rik Farrow

USENIX

The Advanced Computing Systems Association

identifiable fingerprints in network applications

by Jason Damron

Jason Damron has been involved in network security for 10 years. He is currently the lead developer of the Dragon Network Intrusion Detection System for Enterasys Networks.

jdamron@enterasys.com

It is important for both an attacker and a defender to know exactly what is running on a target system. Knowing not only the specific application but also the version number allows one to enumerate the vulnerabilities that are present. Application fingerprinting is a method of determining the type and version of an active network server or client. This is an advanced technique that replaces banner grabbing for situations where a banner doesn't exist or has been removed or obscured.

Typically, banner grabbing consists of initiating a connection to a network server and recording the data that is returned at the beginning of the session. This information can specify the name of the application, version number, and even the operating system that is running the server. Other protocols require that a request first be made, with the resulting response holding the server information.

Savvy administrators have started removing or obscuring this data, making banner grabbing useless or misleading. For instance, an Apache server can be modified to respond with the banner of a Stronghold, an ISS/5.0, or a completely contrived server. Adversaries may then unknowingly choose attacks that will not affect the real Apache server. This method of security through obscurity (while not a good practice by itself) can be a profound gain, since the time it takes to obscure is far less than the time and effort it can take to overcome it.

Application fingerprinting is the field of study that can be used to overcome ambiguity and misdirection. It can be used actively or passively to detect fine distinctions in network programs that give away specific product and version information. Because of this, security through obscurity can definitely increase the challenge but cannot completely remove the threat of a determined attacker's ability to identify the running service. Some have started to undertake the task to counter this "raising of the bar" and are providing tools such as amap and vmap.[1]

1. "The Hacker's Choice – Releases," *http://www.thc.org/releases.php*.

What to Fingerprint?

Any active service on a network can be fingerprinted to some extent. Passively watching the traffic flow by is one method of fingerprinting, but it relies on fingerprints found in normal traffic. This is most efficient when banners are present within the information transfer. However, when banners are not present this method can lose granularity, and it may take a large volume of traffic for the different variations required to precisely determine the application and version information. Active approaches yield much faster results, since the traffic can be intelligently chosen in order to focus the identification process.

Client software can also be fingerprinted. This can be important knowledge in preventing a network breach due to a client-side security flaw. Rogue Internet servers could be loaded with potentially damaging payloads for vulnerable Web browsers, chat clients, media players, or other client programs. These rogue servers could potentially force the traffic of the conversation in a way that will uniquely identify the client. At that point, it could respond with malicious content targeted to the specific client, producing a higher probability of success.

Fingerprinting unknown ports may be somewhat more complicated, because the protocol must be determined first. Banner grabbing can start with the initial return, but if the banner is obscured, deeper inspection must occur. This process may break down to simple trial and error, cycling through the protocols that have been researched.

Even typically passive devices such as an Intrusion Detection System could be identified if set up for active response. An attacker could force traffic to be emitted, triggering a defensive measure such as a TCP session termination. Then by examining the returning traffic for the number of TCP RST packets sent, how fast the sequence numbers grow in attempting to catch up to the ongoing traffic, static values such as IP identity field, and other possible indicators, the fingerprinting software can determine the make and model of the IDS.

Intrusion detection systems have started applying banner grabbing and modeling technologies.

How Is It Possible?

Application fingerprinting works because all actively developed software evolves. New software versions typically include some combination of bug fixes, updates, rewrites, and completely new features. If any of these affect its network presence, it allows the software to be uniquely identified from its predecessors.

The fact that some applications can be uniquely identified by a remote agent is not a vulnerability in the software. In fact, it can be viewed as the opposite, in that vulnerabilities are being removed, which can change the network's characteristics. Ideally, developers who become conscious of this will choose to create a normalized network interface. They could allow modified interactions to take place only when needed by new functionality configured to be active by the user. Alternatively, they could allow configurations that removed unneeded functionality to retain the conventional interaction.

Intrusion Detection

Intrusion detection systems have started applying banner grabbing and modeling technologies. An example of this would be picking out the Sendmail banner and remembering what it is for future reporting, or generating an event immediately if it is known to be vulnerable.

An implementation of application fingerprinting could be the identification of a DNS server from observing a normal DNS conversation. Typical DNS traffic does not include any version information to trigger current systems. However, with careful research it may be possible to detect nuances of the conversation that give away the type and version of the DNS server. At this point, the IDS can respond like any other fingerprinting system – by learning and applying that knowledge to present more intelligent alerts.

Another example could be the security administrator of a large enterprise who doesn't own or operate all of the Web servers. If one of his users makes the effort to obscure the server type and version, they have also obscured it from the IDS. However, if the IDS can monitor enough conversations to determine the type and version, it can then generate events with better information about whether the server was susceptible to the attack.

The above examples are narrow in scope, since they assume a standard port or that the underlying protocol is at least known. A broader version of this idea is an application fingerprinting engine. This engine could be fed by any unencrypted network traffic and first determine the protocol, then move to the type of server/client, and finally, identify the version (if possible). For instance, if a user decided to hide and obscure a Web server on port 44322, an engine such as the one described above could attempt to determine all of the server characteristics to enable the enhanced reporting available for that application.

Case Study: Apache 1.3.x

WHY APACHE?

Apache is the most widely utilized Web server in the world.[2] It is also open source, so code for every version can be downloaded and checked for differences. Previous versions are available at *http://archive.apache.org/dist/httpd/old/*. Other good choices to examine for fingerprints would be BIND and Sendmail. Both are used extensively, and source code for the current and prior versions is freely available. Also, both BIND[3] and Sendmail[4] have had security-related issues in the past, which makes it more important to be able to identify any non-current servers.

Please note that this is not a vulnerability of the Apache Web server. As Apache grows and evolves there will be changes, and some will cause its network presence to change.

2. "Netcraft: Web Server Survey Archives," *http://news.netcraft.com/archives/Web_server_survey.html*.

3. Internet Software Consortium: BIND Vulnerabilities, *http://www.isc.org/products/BIND/bind-security.html*.

4. Vulnerabilities found by searching SecurityFocus vulnerability archive for "Sendmail Consortium," *http://www.securityfocus.com/bid/{2794, 2897, 3377, 3378, 4822, 3163, 5770, 5921, 6548, 5845, 5122, 7614, 7829, 6991, 7230, 8485}*.

TEST SETUP

For this case study, we will assume default installations of Apache servers. The only changes made to the configuration were to make each version listen on a different port for concurrent testing and to start a single instance of the server, since performance will not be an issue. Also, the scope of this research was limited to UNIX versions of the server.

WHERE TO START

The safest and most direct method is to make simple requests. A query for the DocumentRoot can provide useful information such as the options that are present, the order of the options, syntax of the options, and maybe the default Web page, all of which can be used to identify the server. This information is more likely to help determine the type of server rather than an exact version.

Next, simple request errors can be made, such as requesting pages that are not present or leaving out required headers (HTTP/1.1 requires that the Host: header be present). However, because not all Web servers are as robust as Apache, these types of attempts to fingerprint may cause poorly implemented servers to crash.

The accompanying matrix illustrates three simple requests that can be used to narrow down the version of Apache that is being run. They are listed in order of obtrusiveness, starting with the safest method.

Apache Version	TCN Present	Negotiation Options	Bad Method (HEAD 1)
1.3.27	X	Fewer than 24 (missing .lu)	Error 1, 4, 5
1.3.22		Similar to 23	Error 2, 4, 5
1.3 {12, 14, 17, 19, 20, 23, 24, 27}	X	Only 17 and 19 match	Error 1, 4, 5
1.3.11	X	Fewest options	Error 1, 4, 5
1.3 {4, 6, 9}		None	Error 1, 4, 5
1.3.3		None	Error 1, 5
1.3.2		None	Error 3, 5
1.3 {0, 1}		None	Error 1

Returning Error includes:

1. HEAD1 to /index.html not supported.<P>
2. Same as above except URL reads /index.html.en
3. Invalid method in request HEAD1 / HTTP/1.1<HR>
4. Same as above but ends with <P>\n<HR>
5. <ADDRESS> tag present

As can be seen , several of the unique characteristics of a default Apache installation revolve around its ability to negotiate. The first test uses the existence of the Transparent Content Negotiation (TCN) header in any response to divide up the versions. Versions 1.3.11 to 1.3.27 can be broken down further by submitting Accept: statements with an invalid type to force the server to present all known content types. Over time, the supported content types have changed in the default configuration and this can be used to determine the specific versions. The older versions can be broken down further by submitting improper requests and checking for uniqueness in the results. Some of the releases, such as 1.3.{4,6,9}, have very few functional changes in the protocol code, which makes version distinction very difficult. In those cases, fingerprinting may become dependent on the additional modules that have been loaded.

The first step in disguising an application is to enable only the functionality that you require.

RAISING THE BAR HIGHER

The first step in disguising an application is to enable only the functionality you require. The more functionality that is utilized by the network server, the greater the chance for identifying anomalies to be present. Likewise, you don't want to expose yourself to potential security risks that may exist in unnecessary functionality.

The next step is to create a custom configuration to remove as many default responses as possible. Some software does not provide enough flexibility to remove any standard replies. In these cases, either source modifications or binary editing would be required to create the necessary obfuscation. The following are some examples of the options Apache administrators have to increase the complexity required to determine the version information.

Starting with the 1.3.x tree, Apache introduced the ServerTokens directive, which allows the administrator to manipulate the responding Server: header by applying it to the httpd.conf file. However, the version information was always present until version 1.3.12, when it allowed the Prod[uct Only] option, which limits the display to just "Apache."

To further obfuscate the Server: response header, a simple change can be made in the source code. The include file {ApacheTree}/src/include/httpd.h contains #define statements for the product information. The product name can be changed to read:

```
#define SERVER_BASEPRODUCT "GuessMe"
```

Then setting the ServerToken directive to Prod[uct Only] will cause Apache to answer with the following "Server" header:

```
Server: GuessMe
```

The ErrorDocument directive can also be used to deter inquisitive minds from uncovering the type and version of the server. This option, when placed within the httpd.conf file, allows Apache to serve up custom error pages which will remove some of the low-hanging fruit of fingerprinting.

The following is an example of using ErrorDocument to remove identifiers:

```
ErrorDocument 500 /standard-error.html
ErrorDocument 404 /standard-error.html
```

The negotiation alternatives of an Apache server can be modified in the httpd.conf file. When the mod_mime module is enabled, only the languages and character sets that need to be supported should be included, using the AddLanguage and AddCharset directives. Also, if the mod_negotiation module is enabled, the language priority list should also be altered to reflect only languages that are required. The following is a modified section of the httpd.conf file that limits the language priority to only English (en) and German (de):

```
<IfModule mod_negotiation.c>
      #LanguagePriority en da nl et fr de el it ja kr no pl pt pt-br ru ltz ca es sv tw
      LanguagePriority en de
</IfModule>
```

Following the procedures specified above negates many of the fingerprinting techniques discussed and takes a surprisingly small amount of time. Without making these changes, creating fingerprints for individual versions of Apache is a time-consuming task. Spending the extra time to customize the configuration can remove unique characteristics, which causes the application fingerprinting to become much more difficult.

Statement of Ownership, Management, and Circulation, 10/01/03

Title: ;login: Pub. No. 0008-334. Frequency: Bimonthly. Subscription price $110 individuals and institutions. Office of publication: USENIX Association, 2560 9th Street, Suite 215, Berkeley, Alameda County, CA 94710. Headquarters of Publication: Same. Publisher: USENIX Association, 2560 9th Street, Suite 215, Berkeley, CA 94710. Editor: Rob Kolstad. Managing Editor: Alain Hénon, all located at office of publication. Owner: USENIX Association. The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes has not changed during the preceding 12 months.

Extent and nature of circulation	Average no. of copies of each issue in preceding 12 months	Actual no. of copies of single issue published nearest to filing
A. Total no. of copies	8,068	7,550
B. Paid and/or Requested Circulation		
Outside-County Mail Subscriptions	5,223	5,111
In-County Subscriptions	148	0
Sales through Dealers	2,179	1,945
Other Classes	0	0
C. Total Paid and/or Requested Circulation	7,550	7,056
D. Free Distribution by Mail		
Outside County	0	0
In-County	0	0
Other Classes	32	33
E. Free Distribution Outside the Mail	333	400
F. Total Free Distribution	365	432
G. Total Distribution	7,915	7,488
H. Copies Not Distributed	199	515
I. Total	8,068	7,550
Percent Paid and/or Requested Circulation	94%	93%

I certify that the statements made by me above are correct and complete.
Alain Hénon, Managing Editor