The devices create stringency during a normal web server finger printing process. From a pen testing point of view, it is really crucial to detect the device that is acting as a barrier between the source and the target. These devices include load balancers, proxies etc. The target of this paper is to disseminate the HTTP responses and dissect the changed HTTP parameters by intermediate device to trace the actual information about the device. The analysis is based on number of tests conducted during pen testing.

Explanation
The HTTP interfaced embedded devices provide an extra level of security as well as detailed functionality in an infrastructure. If one thinks like a pen tester then these devices try to alter the standard HTTP parameters when it is sent back as a response. The structure of parameter usually gets altered in a HTTP response. On the contrary, it gives a fair idea that an intermediate device was placed on the network. These HTTP devices are very intelligent because the response depends on the type of request and it is necessary to cater to it in an appropriate manner. Once the server responds back to the request, it first goes to the device and necessary alterations are made in the response which is basically the HTTP parameter manipulation. These devices are also highly effective in determining the well formed request and malformed request. The other element which is critical is the type of protocol version supported in a request. The typical requests are HTTP/1.0 and HTTP/1.1. There are certain differences in the way the request is sent to the server. The basic request handling is done by the server itself. Most of HTTP embedded devices work on the concept of “HEADER REWRITING”. In this technique the HTTP headers are rewritten for the response sent by the server.

Working flow of HTTP Request/Response
Let’s have a look at the working flow of HTTP Request/Response Mechanism (see Figure 1).

Technique of fingerprinting:
There are numbers of different methods to follow, but our approach is based on two concepts. Fingerprinting a device which is placed in between a client and server to provide additional functionality has to be traced differentially. Taking this factor into consideration our approach revolves around using understated methods such as:

- Requesting a non existent object from the server.
- Sending a rogue request which is not handled by server.
These techniques look to be an alias of each other but can be used in a different manner when it comes to fingerprinting. It depends a lot on the network architecture too. It requires logic to be placed to gather the information required to fingerprint the intermediate devices. Though you will find strange requests are being sent to server to detect the exception handling and to debug those responses to extract the type of information required.

In order to understand this behavior we will talk about generic concepts of HTTP protocol specification to look at the usage of HTTP parameters.

Connection management is really crucial from bandwidth utilization and request handling from different remote threads. Basically it further requires resources to be provided to the different clients.

The functionality is carried by the connection parameter in the HTTP requests. This works on the concept of forwarding a request from hop to hop and removing the previous header parameters when a request is initiated to next hop.

This actually works on end to end point communication.

There is an inherit functionality or weakness that once the header is forwarded from a device the headers are altered, as only the end point matters irrespective of the path followed. Most devices like load balancers utilize this concept of manipulating the structure of parameters in order to falsify the recipient or are used for spoofing the identity of the internal network from outsiders. But this is

**Listing 1. Google GWS Server Response Via Net Cache Appliance**

Request 1

HEAD /r\n HTTP/1.1
HOST: google.com
HTTP/1.1 301 Moved Permanently
Date: Tue, 09 Dec 2008 06:21:15 GMT
Content-Length: 227
Content-Type: text/html; charset=UTF-8
Expires: Thu, 08 Jan 2009 06:21:15 GMT
Cache-Control: public, max-age=2592000
Server: gws
Location: http://www.google.com/%5Cr%5Cn Via: 1.1 (NetCache NetApp/6.0.6)

Request 2

HEAD /r\n HTTP/1.0
HTTP/1.0 404 Not Found
Date: Tue, 09 Dec 2008 06:21:31 GMT
Content-Length: 1362
Content-Type: text/html; charset=UTF-8
Server: gws
Via: 1.1 (NetCache NetApp/6.0.6)

![Figure 1. Generic HTTP Request Flow Diagram](image-url)
a kind of potential problem which is being exploited by number of HTTP devices used for connection management. Another thing to consider in this is proxies based on HTTP/1.0 do not understand the connection header. As we know in certain requests HTTP/1.1 is backwards compatible with the HTTP/1.0 requests, but in some cases the HTTP/1.1 does exhibit different behavior. If a request is initiated with HTTP/1.0, having connection parameters are ignored because it is considered forwarded in an incorrect manner and hence it is refused.

The Content Length header sets a vector the length of the message in the body. The length is required to understand the bytes used in message transmission when a specific request is initiated. It has been noticed that embedded HTTP devices change the structure of Content-Length. The chunked transfer coding is used to send the message in chunks and pieces rather as full. Again HTTP/1.1 can distinguish easily if a request is being sent by using HTTP/1.0 and no content length is specified. This is real ambiguity as chunked transfers can be truncated and stored as such. This is stringent behavior and that's the reason different HTTP versions work differently when the same request is issued to the server. The device functionality gets changed a lot even when a request is handled with different HTTP versions. Let's see in Listing 1.

In both requests we use carriage return as a requested object from the server. The first request uses HTTP/1.1 and the other is HTTP/1.0. It's necessary to provide a host in the HTTP/1.1 specification. You can deduce that 301 responses are undertaken and are redirected to the location header. This means that the resource in the request is assigned to a new URI which is specified in the location header. The second request is made with HTTP/1.0 and straight forward 404 responses are provided as the output. This field specifies that the request is initiated through the proxy with no modification in the parameters. One can see the behavior of the server when a different HTTP specification is used. On the contrary, by exchanging HTTP specifications, a different pattern of information can be extracted.

The proxies and intermediate gateways use the VIA HTTP parameter for forwarding request to destination servers. The presence of the VIA field simply projects that the gateway or proxy is placed as an intermediate device to forward a request. The response field in the VIA header carries certain comments which reflect the type of software used by the proxy or the gateway analogous to the user agent and server header fields. The comments should be removed to display less information of the intermediate device. It is considered as good practice because unnecessary comments in the request leverage a lot of information (see Listing 2).

The comments in the bracket project that a Net Cache device is being used to forward this request.

The status code of 305 always reflects that the proxy should be used for this specific response. This means the resource should be accessed through the proxy provided by the location http header. This is one of the most critical

---

**Listing 2. HTTP request through VIA HTTP Header**

```
HEAD / HTTP/1.1
HOST: yahoo
HTTP/1.1 200 OK
Date: Tue, 09 Dec 2008 07:29:02 GMT
Content-Type: text/html
Connection: close
Server: Apache/1.3.33 (Debian GNU/Linux)
Via: 1.1 cac01 (NetCache NetApp/6.0.6)
```

**Listing 3. Citation 1**

```
$ ./nc www.example1.com 80
HEAD /r/n HTTP/1.0
HTTP/1.0 302 Object Moved
Date: Tue, 09 Dec 2008 16:37:20 GMT
Server: NS8.0.48.7
Content-Type: text/html
Cache Control: private
Location: www.example.com/index.html
$ ./nc www.example2.com 80
HEAD /r/n HTTP/1.0
HTTP/1.0 302 Object Moved
Date: Tue, 09 Dec 2008 16:49:10 GMT
Server: NS3.0
Content-Type: text/html
Cache Control: private
Location: http://www.example2.com /on
```
aspects from a security perspective because it is hard to distinguish whether the request originated from an authorized server or not. This functionality is also used by embedded HTTP devices in conjunction with other techniques.

The status code 403 forbidden and 404 objects not found are one of the standard responses required to fingerprint devices placed in the path. As discussed above a request is always issued for the resource which does not exist to check the status and response from the server.

Another point to look into is caching. The intermediate devices functionality depends on the process of caching when it comes to request handling. The caching is not considered semantically transparent because the responses get altered as per the specification of direct communication between the client and the server. It means the response thrown back to the client have a different HTTP header layout as per the standard response returned from the server. Caching depends a lot on the HTTP specification used. Basically, a heuristic approach is used to cache a response. Depending on the improper response that is cached, this will impact the performance to some extent. An entry can be treated as fresh if it is not expired and considered as stale if expired. The above caching is also used by embedded HTTP devices and a lot of alterations can be made in the response sent back.

These are the standard HTTP considerations that need to be taken into account while finger printing embedded HTTP devices.

### The Probable Detection Points

There is certain specific functional behavior that is shown by various HTTP devices and the way HTTP headers are handled.

Embedded HTTP devices definitely change the layout of HTTP headers. One of the processes followed by these devices is HTTP Header Breaking. In this technique, the headers are broken and the tester will find a different order of headers in a response.

This indicates directly that an intermediate device is playing a trick and the tester has to analyze this behavior by dissecting the HTTP responses in a detailed manner.

The basic signatures are:

- **Content-Length** > Content Length
- **Content-Type** > Content Type

Embedded HTTP devices also use the technique of HTTP Header Manipulation. In this technique, HTTP headers are manipulated from normal structure to a different layout. This indicates directly about the presence of certain devices in between the client and the server.

The basic signatures are:

- **Content** > Content
- **Connection** > close

---

#### Listing 4. Citation 2

```bash
GET /nc www.example1.com 80
HEAD /rl\n HTTP/1.0
HTTP/1.0 404 Not Found
Content-Length:
Server: Apache
Content-Type: text/html; charset=iso-8859-1
Accept-Ranges: bytes
Cache-Control: no-cache, no-store
Content-Length: 329
Connection: close
```

#### Listing 5. Citation 3

```bash
GET /nc www.findstone.com 80
HEAD /rl\n HTTP/1.0
HTTP/1.0 302 Object moved
Date: Tue, 09 Dec 2008 16:52:58 GMT
Content-Length: 121
Content-Type: text/html
Expires: Tue, 09 Dec 2008 16:52:58 GMT
Cache-Control: private
Server: Microsoft-IIS/5.0
X-Powered-By: ASP.NET
Set-Cookie: ASPSESSIONIDDAQQMAAG=NOHL1GEDKBHMMOBRKAMAMHF; path=/
Set-Cookie: BIGipServerhttp.pool=1562574858.20480.0000; path=/
Location: http://www.findstone.com
```

#### Listing 6. Citation 4

```bash
GET /Actions/DescribeImagesAWSAccessKeyId=0C2QCKRKS3J69PF56QQR2&Owner.1=084307701560SignatureVersion=14&Timestamp=2007-02-15T7%3A30%3A13 4Version=2007-01-03Signature<v

9PZ6QQQR24ImageId=ami-00b95c69&OperationType=addsSignatureVersion=14
Timestamp=2007-02-15T7%3A30%3A14&OwnerGroup.1=ali1Version=2007-01-03Signature<v

reply: HTTP/1.1 200 OK\r\n'header: Server: Apache-Coyote/1.1
header: Transfer-Encoding: chunked
header: Date: Thu, 15 Feb 2007 17:30:13 GMT
send: GET/?Action=ModifyImageAttribute&Attribute=launchPermissionAWSAccessKeyId=0C2QCKRKS3J6
```
This behavior can be implemented on other HTTP headers too.

The third resultant factor is HTTP Header Reordering. Usually web servers like IIS and Apache have set hierarchy of displaying HTTP headers. The devices sometimes change the normal pattern of HTTP headers to something different. Basically the reordering is accompanied with header manipulation and broken headers.

It gives a straightforward idea about the HTTP device being placed in the route.

Another point of consideration is the IP based session management by some of the devices. These devices use the HTTP pool parameter for doing session management.

The set-cookie parameter reflects this behavior. It can be possible that the set-cookie header have the commented name of the HTTP embedded device or not. But this is a certain technique used for fingerprinting the device that perform IP based session management.

These are the generic concepts that can be used in determining the HTTP based devices.

**Practical Citation of HTTP Embedded Device Fingerprinting**

After discussing core concepts we will look into certain examples to understand the issue in overall perspective. The devices will be traced based on the responses (see Listing 3).

Explanation: If you dissect this response you will find that the content type and cache control parameter is altered and is broken from the normal layout.

This time the server string is present with NS 8.0.48.7 and ns_3_0.

The presence of the server string gives an idea, but broken HTTP headers ensure that this is a NET SCALAR device (see Listing 4).

Explanation

If you dissect this HTTP response you will see content is manipulated to Xontent. The Apache web server does not show this type of strange behavior from any perspective. This ensures an embedded HTTP device is placed in between the path. This type of behavior is shown by RADWARE devices in a real world application (see listing 5).

Explanation

This time not alteration in HTTP headers but Set-Cookie I set with HTTP pool parameter and the device string is passed directly. This straightforward reflects a Big IP device (see Listing 6).

Explanation

If you dissect this request you will find the connection parameter is changed to nnCoection. This type of behavior is shown mostly by Net Scalar devices.

**Conclusion**

The probabilistic approach is a good element of fingerprinting devices in a number of situations. It has been the type of concepts used embedded HTTP devices and the way operations are performed on HTTP headers. The applicability of these techniques reasonably ensure the presence of devices and to some extent the type of device. At last we want to discover the hidden device in the path and through these techniques it is quite efficient. That’s the way it is.

**Aditya K Sood**

Aditya K Sood is an Independent Security Researcher and Founder of SecNiche Security. He is a Lead Author for Hackin9 group for writing security and hacking papers. His research has been featured in Linux, login magazine and Elsevier Network Security Journals. Aditya’s academic background holds a BE and MS in Cyber Law and Information Security from Indian Institute of Information Technology (IIIT-A). He had already spoken at conferences like EuSecWest, XCON, Xiugfng, OWASP, ClubHack, CERTIN etc. In addition to that he is a team lead at Evilfingers community.

His other projects include Mtabs, CERA and Triosec. He has written number of security papers released at packetstorm security, Linux security, infosecwriters, Xssed portal etc. He has also given number of security advisories to forefront companies.

At present he is working as a Security Auditor in KPMG IT Advisory Services where he handles large scale security assessments project.