

Technical Note by Amit Klein: Detecting and Preventing HTTP Response Splitting and HTTP Request Smuggling Attacks at the TCP Level

Source: <http://www.derkeiler.com/Mailing-Lists/securityfocus/bugtraq/2005-08/0201.html>

From: Amit Klein (AKsecurity) (aksecurity_at_hotpop.com)

Date: 08/15/05

Date: Mon, 15 Aug 2005 20:31:00 +0200
To: bugTraq <bugtraq@securityfocus.com>

Technical Note: Detecting and Preventing HTTP Response Splitting
and HTTP Request Smuggling Attacks at the TCP Level

Amit Klein, August 2005

Introduction

=====

This technical note describes a detection/prevention technique that works in many cases both with HTTP Response Splitting and with HTTP Request Smuggling. This technique makes use of implicit information found in the TCP stream, namely the segmentation into packets and the TCP PSH bit. In HTTP Response Splitting, this technique needs to be applied at the proxy server, the one closest to the web server, and to the response stream. In HTTP Request Smuggling, this technique needs to be applied at the entity closest to the attacked proxy server/device (i.e. implemented in another proxy server, or the web server itself), and to the request stream (note, however, that this second server may be off the premises of the organization wherein the web server is, see also "Can HTTP Request Smuggling be blocked by Web Application Firewalls?", <http://www.securityfocus.com/archive/107/402974>).

TCP PSH bit and PUSH flag

=====

Before describing the technique, the reader is reminded what are the TCP PSH bit and PUSH flag. RFC 793 (Transmission Control Protocol, <http://www.ietf.org/rfc/rfc793.txt>) defines the push functionality as following (from section 1.5):

Sometimes users need to be sure that all the data they have submitted to the TCP has been transmitted. For this purpose a push function is defined. To assure that data submitted to a TCP is actually transmitted the sending user indicates that it should be pushed through to the receiving user. A push causes the TCPs to promptly forward and deliver data up to that point to the receiver.

And in section 2.8:

The sending user indicates in each SEND call whether the data in that call (and any preceding [sic] calls) should be immediately pushed through to the receiving user by the setting of the PUSH flag.

This is realized by a PSH bit in the TCP header.

RFC 793 also mandates that the socket API provides means for the caller to set this bit via a PUSH flag in the SEND function, and to likewise receive it in the RECV function.

However, this requirement was later waived in RFC 1122 (Requirements for Internet Hosts – Communication Layers, <http://www.ietf.org/rfc/rfc1122.txt>), section 4.2.2.2:

A TCP MAY implement PUSH flags on SEND calls. If PUSH flags are not implemented, then the sending TCP: (1) must not buffer data indefinitely, and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent).

The discussion in RFC-793 on pages 48, 50, and 74 erroneously implies that a received PSH flag must be passed to the application layer. Passing a received PSH flag to the application layer is now OPTIONAL.

And indeed, the two implementations of sockets, the UNIX BSD sockets and WinSock, do not implement the PUSH flag neither in SEND nor in RECEIVE:

*) FreeBSD (example of standard BSD sockets)

send(2) man page:

<http://www.freebsd.org/cgi/man.cgi?query=send&apropos=0&sektion=2&manpath=FreeBSD+5.4-RELEASE+and+Ports&format=html>

recv(2) man page:

<http://www.freebsd.org/cgi/man.cgi?query=recv&apropos=0&sektion=2&manpath=FreeBSD+5.4-RELEASE+and+Ports&format=html>

*) Sun Solaris 8 (another example of standard BSD sockets)

send(3SOCKET) man page: <http://docs.sun.com/app/docs/doc/806-0628/6j9vie803?a=view>

recv(3SOCKET) man page: <http://docs.sun.com/app/docs/doc/806-0628/6j9vie7u0?a=view>

*) Microsoft Windows WinSock (2.0) sockets:

send:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/send_2.asp

recv:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winsock/winsock/recv_2.asp

This means that according to RFC 1122, these implementations should simply set the TCP PSH flag in the last TCP segment of the caller's data buffer. Unfortunately, this also means that socket applications do not have direct access to the PSH bit (in other words, it would take some hacking around the standard sockets API to get this information – thus making the whole idea presented below much harder to implement in pure sockets applications). To illustrate: a sender uses the sockets send() function to send a payload of 2000 bytes over a TCP circuit on a LAN. The call is (assuming C API):

```
send(socket, buffer, 2000, ...);
```

This results in 2 TCP segments (IP packets), the first carrying a payload of 1448 bytes (maximal LAN packet, minus level 2, IP and TCP headers), the second carrying a payload of the remaining 552 bytes, with a PSH bit set.

In contrast, having the caller invoke send twice to send the same data:

```
send(socket, buffer, 1000, ...);  
send(socket,buffer+1000, 1000, ...);
```

Results in 2 TCP segments, both having 1000 bytes of payload, and both having the PSH bit set.

Therefore, by inspecting the TCP header, one can figure out (in this rather simplistic example) whether the data was sent via a single call to send(), or via two calls.

This observation is the basis for the following technique described below.

Detecting HTTP Response Splitting and HTTP Request Smuggling

Detection of HTTP Response Splitting attack can be realized by fulfilling the following requirements (the technique can be applied to HTTP Request Smuggling with some obvious modifications):

Requirement #1: Deny "pipelined" traffic, that is, do not accept a second response before a first response was fully served, and a second request was fully received. Particularly, a packet containing data for the first response must not contain superfluous data beyond the end of the first response (i.e. a second response).

Requirement #2: Since from #1 it follows that the end of the first response coincides with an end of a packet, and end of transmission, such packet should contain a PSH bit set (see above).

Requirement #1 is obvious, yet insufficient, as shown in the successful HTTP Response Splitting attacks against Squid and NetCache (see "Divide and Conquer – HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics", http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf, pp. 15–19), although both products impose packet boundary requirements (making sure that the first response terminates at a packet boundary isn't so hard if the packet length is known to the attacker). But if we are to add requirement #2, the attack can be thwarted in many cases, because the attacker has no way of forcing the web server to send the PSH flag (remember that from the web server's perspective, it is in the middle of the first response, and will send the PSH bit only at the end of the first response). Of course, it may be possible that the web server always sends the PSH bit (for example, the TCP stack of IBM TPF 4.1 apparently does so by default – see the description of TCP_PSH_LAST ioctl option in <http://www-306.ibm.com/software/http/tpf/serv/GTPMLB19.pdf>), or sends it after each HTTP response header, or in the middle of the response, but that is not the case with many servers (although I have seen servers that on occasion will send the PSH flag in the middle of HTTP responses).

So, while it may not be perfect, it seems that this technique has a high detection probability (much higher than enforcing requirement #1 alone), yet very low likelihood for false positives.

Notes

=====

1. The technique fails if there are TCP/HTTP aware devices between the web server and the proxy server, as these may alter the TCP stream.
2. The technique can be applied to HTTP Request Smuggling as following: the proxy server forwards the requests to the web server. The web server (or the receiving entity) needs to verify that the request ends on a packet boundary, and that the PSH flag is set.
3. While the technique covers detection of attacks, since such detection is carried out in real-time, it is possible to terminate the TCP connection (or perform other actions) and thereby to block the attack.
4. This technique lends itself nicely to detection/prevention by network IDS/IPS, as it only requires sniffing the TCP/IP traffic and flagging HTTP requests/responses that do not terminate on a

packet boundary, with PSH bit set.

Alternatives

=====

Other, complementary methods (for prevention) that are known to work are:

1. The web site can use SSL connections (HTTPS) only. This will only eliminate 3rd party proxy servers. It does not eliminate the browser cache issue, and it may not handle the site's own cache server (or any other on-site HTTP aware devices). Of course, migrating to SSL only has many real-life drawbacks.
2. A proxy cache server can be configured not to use persistent HTTP connections with the *server*, in which case, its cache will not be poisoned. This prevention technique has significant performance impact.

Summary

=====

Observing that an end of HTTP request/response message is aligned with a TCP segment (packet) boundary, *and* that that segment has the PSH bit set, can be an effective way to detect and prevent HTTP Response Splitting and HTTP Request Smuggling. This technique can be easily employed when the TCP layer is directly accessible (as opposed to the sockets model).

Off topic personal notice/clarification

=====

I am often asked whether I work for Watchfire, or did so in the past. So let me state the following:

1. I am not employed by Watchfire, and I am not part of any Watchfire team.
2. In fact, I have never been employed by watchfire (I was employed by Sanctum for many years, but I quit Sanctum slightly before it was acquired by Watchfire).

Having said that, I have nothing against Watchfire, and I wish them best of luck.