SYN-Floods and Countermeasures-Part 2

David Holmes, 2012-26-06

Last week we looked at the mechanics of the SYN-flood attack. Now let's examine the mechanics of the countermeasures.

Countermeasure: SYN-Cookies

In 1996, Dan Bernstein came up with an elegant defense against the SYN-flood called SYN-cookies. SYN-cookies enable the server to respond to all SYN packets while only creating flow table entries for legitimate connections.

Figure 2: Animation – SYN floods and SYN cookies

The SYN-cookie does this by encapsulating three fields of the client's SYN packet into a 32-bit value. The value contains just enough information about the original SYN packet that the server needs to know later for creating a flow table entry. The value is encrypted and this ***cookie*** is sent back to the client in the SYN-ACK packet. The important thing to understand is that the SYN-cookie allows the server to respond to the initial SYN ***without*** creating a flow-table entry.

Legitimate clients will send the cookie back to the server with its final ACK packet. The server can then decode the cookie, and using the information within, create the flow-table entry. Neat! The encrypted values of the three fields, which are carried back to the server in the SYN-ACK, make this possible.

Packets that are really just SYNs from a SYN-flood will never return the cookie (because they never send back the final ACK) and thus, will never eat up flow table space.

SYN-Cookie shortcoming: not all valid connections can be sustained

While SYN-cookies protect the flow-table from filling up with invalid connections, there is a drawback to the original SYN-cookie algorithm. Remember how the server only encodes three fields of the original TCP SYN packet? It turns out the other fields that were lost are handy fields after all, at least in certain circumstances covered below.

Countermeasure: SYN-Cache

Because of these shortcomings, the Linux community briefly disabled SYN-cookies and experimented with a different approach in 2008 called the SYN-Cache. A miniaturized flow table entry was created when the initial SYN was received. It was hoped that this smaller initial cache would enable the tables to grow large enough sustain services while under SYN-flood attack.

SYN-Cache shortcoming: results mixed

SynCache performance results are mixed, depending on which data that you look at. The 2008 Linux effort showed that the connection table was optimized for size to take into account SYN-floods but empirical testing suggested that the hosts were quite vulnerable even to small SYN-floods. A trivial SYN-flood from a single host appeared to be enough undermine this approach.

> Ross Vandegrift [Linux]: Under no SYN flood, the server handles 750 HTTP requests per second, measured via httping in flood mode. With a default tcp_max_SYN_backlog of 1024, I can trivially prevent any inbound client connections with 2 threads of SYN flood. Enabling tcp_SYNcookies brings the connection handling back up to 725 fetches per second.

A FreeBSD study had a different conclusion. According to the empirical testing in the FreeBSD paper, SYN-Cache performance was on par with SYN-Cookies.

> "[T]he syncache is effective at handling a SYN flood while still allowing incoming connections. Here, 99% of the incoming connections are completed within 300 microseconds, which is on par with the time required to connect to an idle unmodified system."

Countermeasure: SYN-Proxy

A typical conventional firewall is a pass-through device – meaning that traffic comes into the device, is inspected, and then either dropped or passed through to the server. Previously pass-through was thought to be the fastest way to process traffic, but it also makes implementing SYN-cookies difficult as it requires the firewall to be stateful. When a pass-through firewall detects a SYN-flood, it switches into SYN-proxy mode and temporarily stalls TCP connections before sending them through. In SYN-Proxy mode, the firewall, upon receiving the SYN, will "hold" the connection by responding to the client with a SYN-ACK and then wait for its ACK. If the client never sends the ACK, the connection is eventually timed out. If the client does answer back, the SYN is replayed to the server behind the firewall. The firewall patches up the connection between the client and the server.

SYN-Proxy shortcoming: resource exhaustion at the firewall

The problem with the SYN-Proxy approach is that enough SYNs directed to a firewall can fill its connection table just as they can for a server. As we've seen in the SYN-flood description, a full connection table results in a reboot or a denial of service.

Countermeasure: TCPCT - TCP Cookie Transactions

If the problem with TCP is that a client can force a server to create flow table entries, why not change the TCP protocol itself to allow for behavior similar to SYN-cookie while still retaining all the packet field data that was lost when SYN-cookies were used?

That's the proposal of RFC 6013 – "TCP Cookie Transactions". It is a modification of the TCP Protocol itself to enable a connection to be forged in an entirely stateless fashion. It also purports to strengthen TCP against other attacks, such as forged IP addresses, predictable ports, discoverable sequence numbers and malicious resets.

TCPCT would have both the client and server pass cookies to each other during a new four-way handshake. The paired cookies in TCPCT, in theory, will provide protection at least as good as traditional SYN-cookies.

TCPCT shortcoming: client support

The main drawback to TCPCT is that the lack of support for it among clients. Only recent Linux kernels support it, and most clients these days are still Microsoft Windows or BSD (Apple)-based. What made SYN-cookies so successful was the fact that they didn't require a change to the clients – it was a neat server-side only tweak. This isn't the case with TCPCT. Because TCPCT is a fundamental change to the TCP protocol, it must be supported by the client and the server (or any middleboxes or proxies). Even if a vendor were to offer some kind of TCPCT -> TCP proxy for servers, it still wouldn't see much use until a tipping point of clients supported it as well.

TCPCT shortcoming: DNSSEC or not

In the abstract for RFC6013, the authors state that a driving force for TCPCT is the adoption of DNSSEC. DNS is the domain name system for the Internet, which maps names like www.example.com to addresses like 226.174.183.229 and nearly all clients rely on the global DNS system to make the Internet work. It's a relatively simple protocol based on stateless UDP (as opposed to the stateful TCP). A secure version of DNS (called DNSSEC) is rolling out across the Internet now. Because DNSSEC packets are substantially larger than DNS packets, there is talk of switching to TCP from UDP. If this happens, the authors pose, SYN-floods against DNSSEC servers will impact name services, making DNSSEC vulnerable to the SYN-floods where it was not before (DNS was typically stateless).

However, there is a fundamental problem with this argument. The missing fields that were not encoded in the original SYN-cookie algorithm are related to long-lived, high-bandwidth connections. DNSSEC transactions are not long-lived; therefore avoiding SYN-cookies in a DNSSEC environment isn't really a valid driver.

One situation that might cause adoption of TCPCT is if SYN-flood attacks jump up by an order of magnitude and become so widespread that clients and servers ***have*** to implement it in high-speed, high-bandwidth environments.

Conclusion

So what is everyone using today? Some conventional firewalls are still using SYN-proxy, or a combination of SYN-proxy and SYN-cookies. Other devices, such as Linux and BSD-based servers employ threshold-activated SYN-cookies. That is, they will typically run ***without*** SYN-cookies enabled, making full use of all the standard TCP fields. When they detect that they are under attack, usually by seeing a spike in the number of SYNs they switch to SYN-cookies. When the attack appears to be over, they'll switch off SYN-cookies and resume normal TCP handshakes.

SYN-cookies are likely the best defense that we'll have in the short and medium term. High-capacity perimeter devices (like, ahem, F5's BIG-IP family) employ specialized hardware to accelerate the SYN-cookie defense.

The longevity of the SYN-flood attack suggests that it will probably be with us as long as TCP exists in its current form and TCP isn't going away anytime soon. Web applications using HTTP, JSON, AJAX (and the like) rely on TCP and that's unlikely to change. Distributed denial-of-service attacks aren't going way, either – if anything, they are increasing and the SYN-flood is a cheap and easy way for attackers to wreak havoc on the unprepared.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com