Mitigating the Apache Killer Vulnerability

David Scholberg
CS Master's Project

What is the Apache Killer?

- A vulnerability involving how Apache processes HTTP range requests
- A specially crafted HTTP request can cause a DoS attack through excessive CPU and RAM usage
- Versions 2.2.19 and prior are vulnerable

An example HTTP header

GET / HTTP/1.1
Host: www.google.com
Range: bytes=12-294,150-304
Accept-Encoding: gzip
Connection: keep-alive

HTTP header that may kill Apache

GET / HTTP/1.1

Host: www.google.com

Range:bytes=0-1,0-2,0-3,0-4,0-5,0-6,0-7,0-8,0-9,0-10,0-11,0-12,0-13,0-14,0-15,0-16,0-17,0-18,0-19,0-20,0-21,0-22,0-23,0-24,0-25,0-26,0-27,0-28,0-29,0-30,0-31,0-32,0-33,0-34,0-35,0-36,0-37,0-38,0-39,0-40,0-41,0-42,0-43,0-44,0-45,0-46,0-47,0-48,0-49,0-50,0-51,0-52,0-53,0-54,0-55,0-56,0-57,0-58,0-59,0-60,0-61,0-62,0-63,0-64,0-65,0-66,0-67,0-68,0-69,0-70,0-71,0-72,0-73,0-74,0-75,0-76,0-77,0-78,0-79,0-80,0-81,0-82,0-83,0-84,0-85,0-86,0-87,0-88,0-89,0-90,0-91,0-92,0-93,0-94,0-95,0-96,0-97,0-98,0-99,0-100,0-101,0-102

Accept-Encoding: gzip

Connection: keep-alive

The vulnerability

- Vulnerable versions of Apache process overlapping ranges very inefficiently
- An HTTP header with too many overlapping ranges will cause a DoS through RAM and CPU consumption
- Such an HTTP header is fairly trivial to construct and send

The fix

- Create an Intrusion Prevention System (IPS)
- IPS is a proxy server in front of Apache
- Traffic destined for Apache goes to the proxy server first
- Proxy server analyzes HTTP header; if too many overlapping ranges, block connection and log IP address

Caveat

- Apache Killer was fixed very quickly
- Has long since been fixed
- Apache Killer is used to illustrate the process of creating an IPS for application level exploits

IPS Design

IPS is a special Intrusion Detection System (IDS) that detects and blocks attacks

Must be in between client and server to block attacks

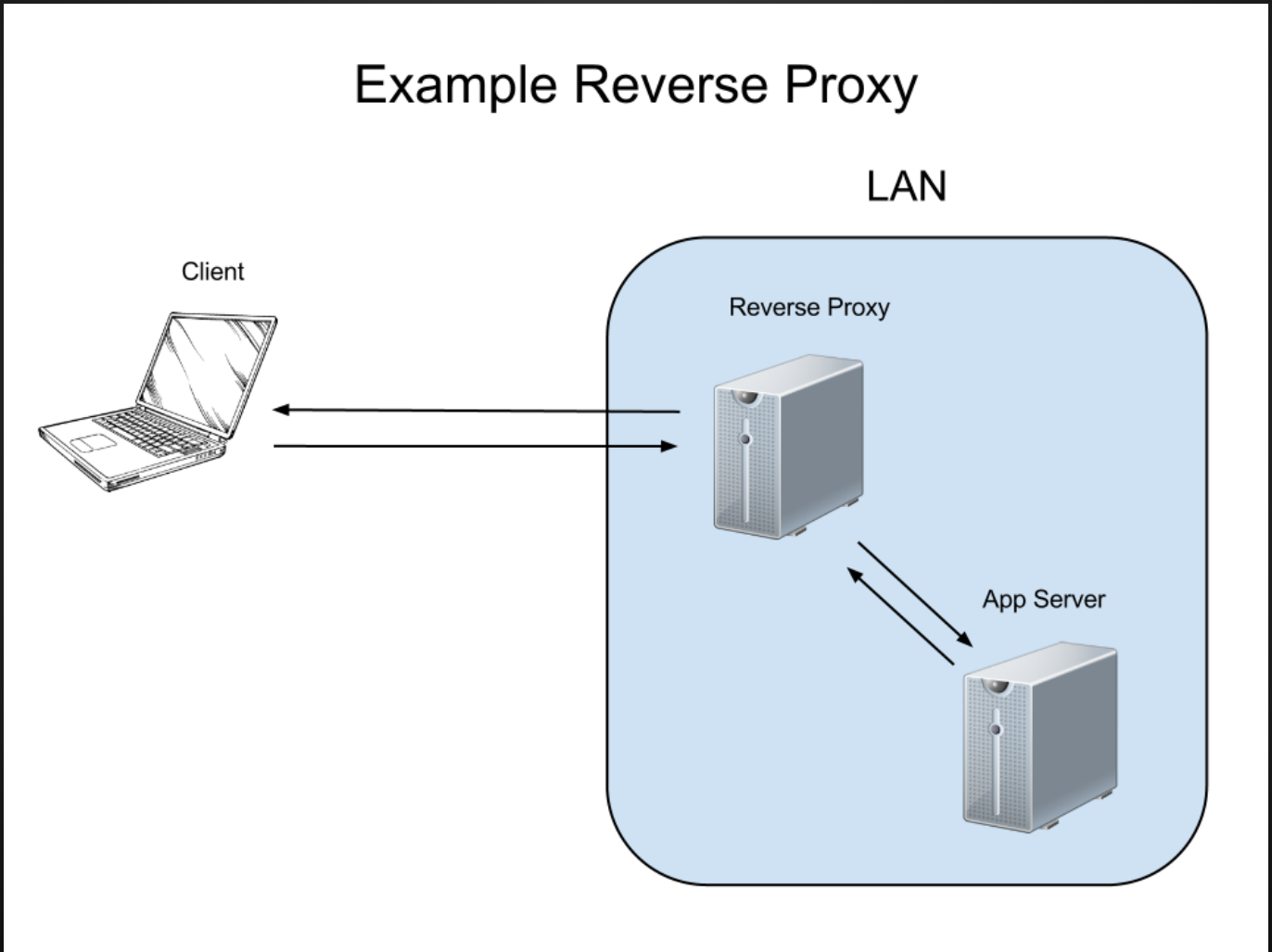
Can operate on any level of network stack, and usually multiple levels

Reverse Proxy

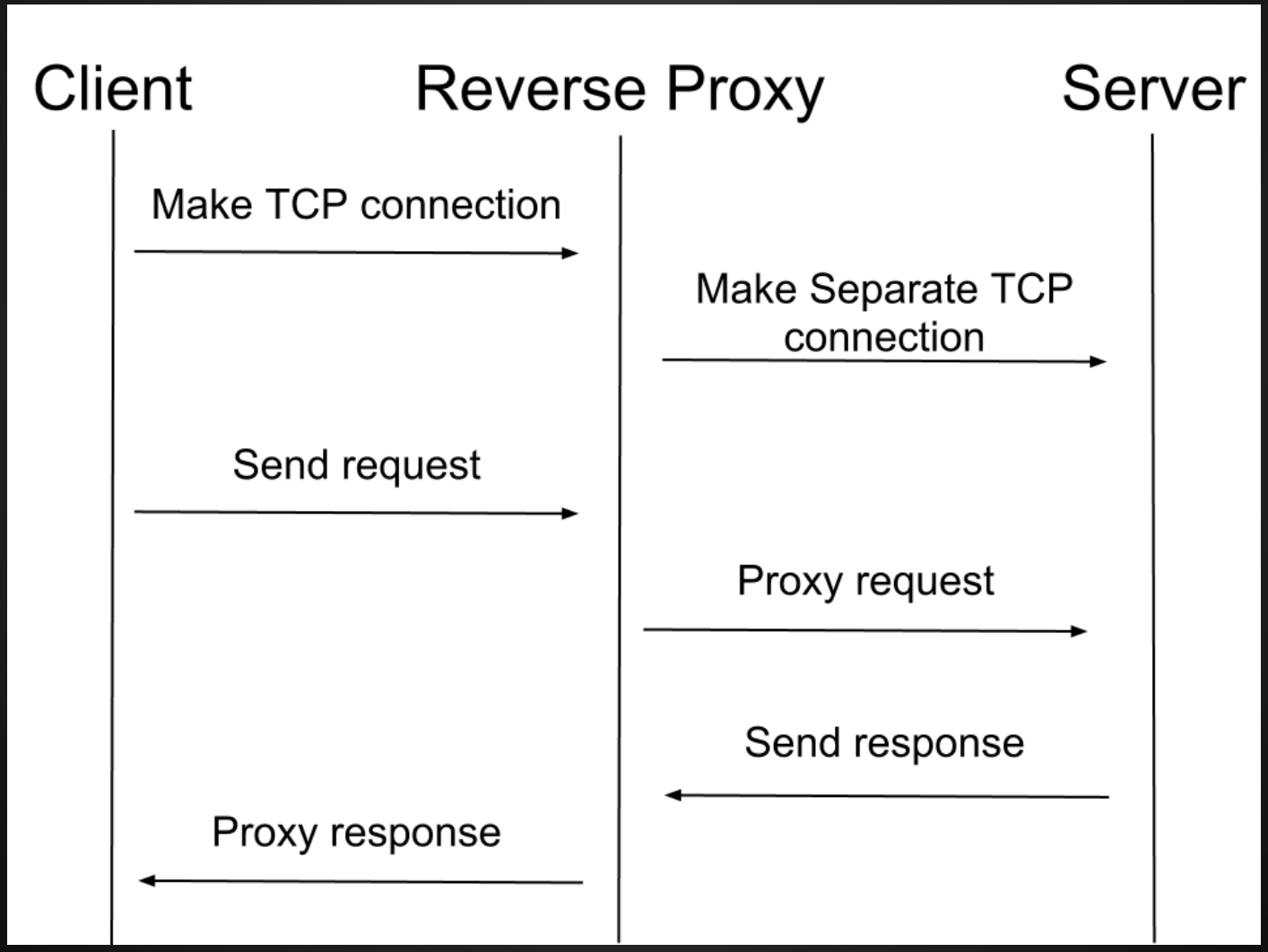
IPS will be designed as a reverse proxy

Reverse proxy is a server that proxies traffic unbeknownst to the client

Reverse Proxy



Reverse Proxy



Reverse Proxy Pseudocode 1

```
while true
   read from client socket into free space in client buffer
   if more client data was received
       determine fate of client data (allow, buffer, or block)
       if client data is blocked
           exit loop

   read from server socket into free space in server buffer
   if more server data was received
       determine fate of server data (allow, buffer, or block)
       if server data is blocked
           exit loop
```

Reverse Proxy Pseudocode 2

```
    if there is client data to send and data is not to be buffered
        send client data to server
        if not all data was sent
            shift remaining client data to start of client buffer

    if there is server data to send and data is not to be buffered
        send server data to client
        if not all data was sent
            shift remaining server data to start of server buffer

end while
```

Apache Killer IPS

After client data is received, but before it is sent along to the server, it is examined

If data from the client contains too many ranges, drop data and close connections

Data from the server is implicitly trusted

Apache Killer IPS

Basic algorithm for attack detection:

```
set proxy flag to allow

if client data doesn't contain "\r\n\r\n"
    set proxy flag to buffer
else if client data contains valid HTTP declaration
    if client data has a range section
        if number of ranges exceeds threshold
            set proxy flag to block and log client IP
```

Implementation

Written in C for GNU/Linux

Linux socket library used for TCP connections

Perl Compatible Regular Expressions (PCRE) library used for pattern matching

Overview of Linux socket programming

A listening socket can be created with calls to `socket()`, `bind()`, and `listen()`

Client connections can be accepted by calling `accept()` on the listening socket, which will return new socket connected to client

The new socket can be used for read/write operations to communicate with client

Overview of Linux socket programming

To initiate a connection with a server, call `socket()` and `connect()`

Forking

```
// Create child process to handle client connection.
// The return value of fork tells the process whether it is the
// parent or the child.

int pid = fork();
if(pid == 0) {
   // this is the child process
   handle_client_socket(
       client_socket,
       inet_ntoa(client_addr.sin_addr)
   );
}
else if (pid < 0) {
   // fork() returned an error
   die("fork() failed");
}
else {
   // this is the parent process
   close(client_socket);
}
```

Regular Expressions

```
const char *regex_strings[REGEX_NUM] = {
   "\r\n\r\n$",
   "HTTP\/[0-9]+?\.[0-9]+?[^0-9]",
   "Range:.+?=(.*?)\r",
   ",?[0-9]*?\-[0-9]*"
};
```

Logging with syslog

Setup logging

```
openlog("reverse_proxy", LOG_PID, LOG_USER);
```

Write log entry

```
syslog(
    LOG_WARNING,
    "data from %s was rejected\n",
    client_addr
);
```

Testing Environment

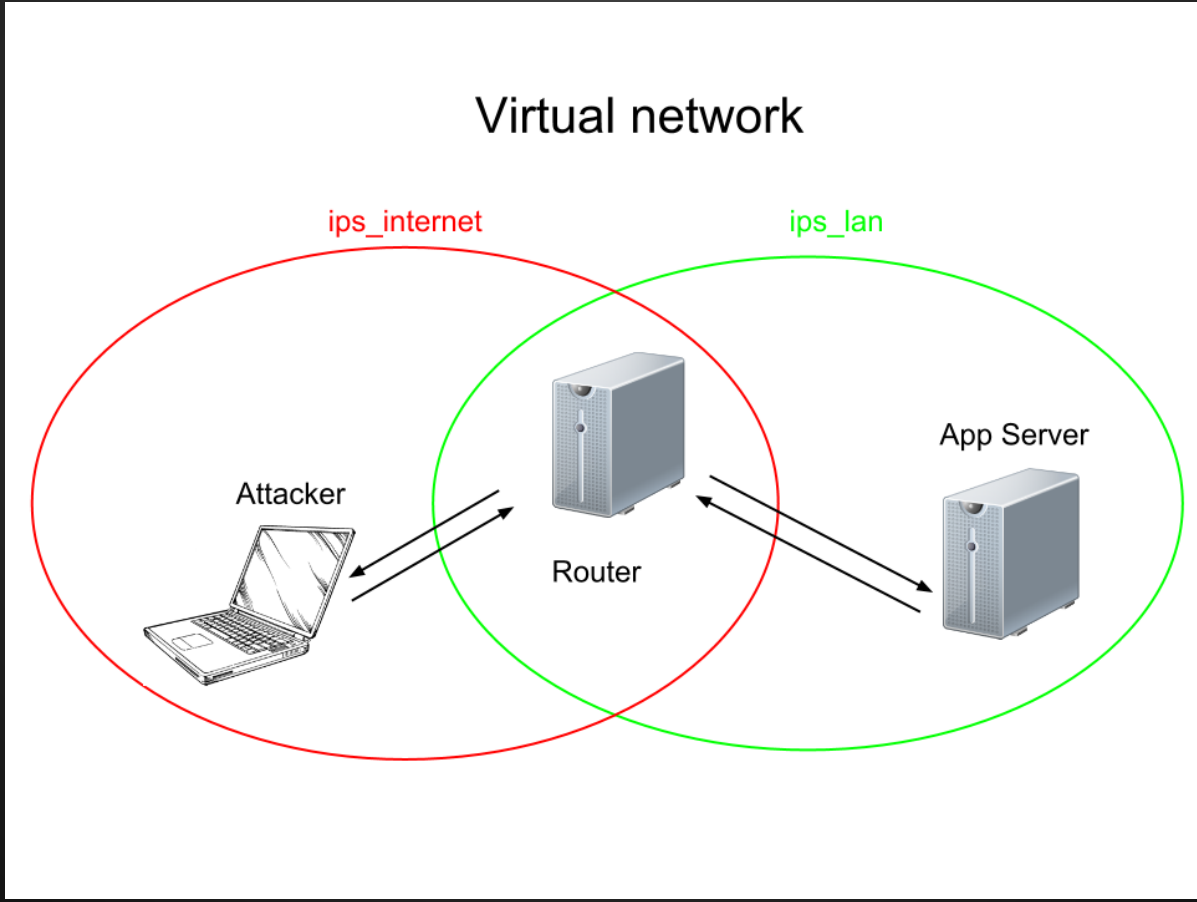
3 virtual machines (VMs), created with VirtualBox: Attacker, Router, and App Server

All 3 VMs run CentOS 6.3

The Router is on two virtual networks. The attacker is on one of those virtual networks, and the App Server is on the other.

Testing Environment

Network of VMs



Testing Environment

VM 1

Name :	Attacker
OS:	CentOS 6.3
RAM:	1024 MB
CPUs:	1
Internal Network:	ips_internet (IP: 7.0.0.2)
Software:	Metasploit, killapache.pl, w3m

Testing Environment

VM 2

Name :	Router
OS:	CentOS 6.3
RAM:	512 MB
CPUs:	2
Internal Network:	ips_internet (7.0.0.1), ips_lan (10.0.0.1)
Software:	iptables, Apache Killer IPS

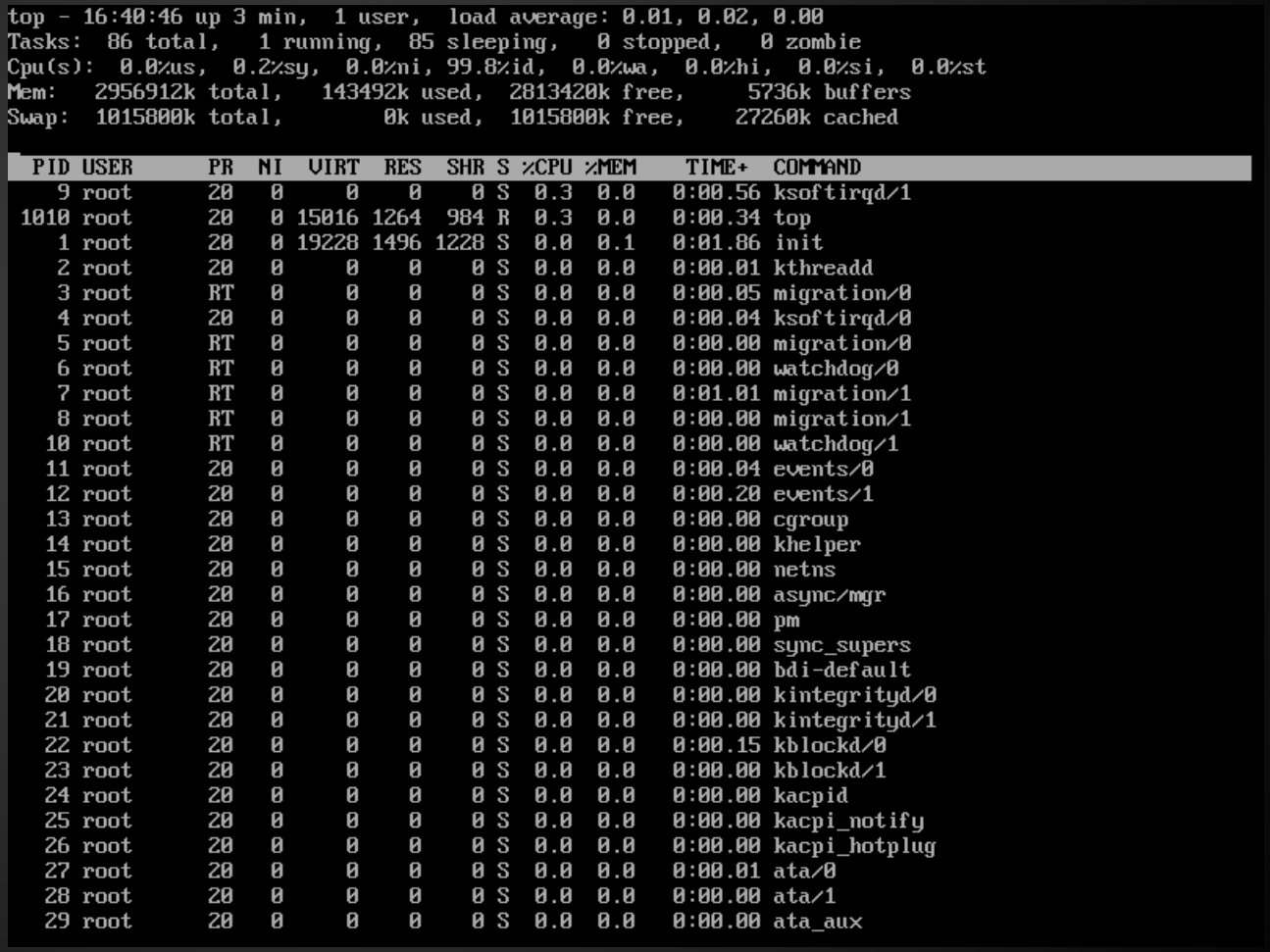
Testing Environment

VM 3

Name:	App Server
OS:	CentOS 6.3
RAM:	3072 MB
CPUs:	2
Internal Network:	ips_lan (IP: 10.0.0.2)
Software:	Apache 2.2.18

Results - No IPS

Before Attack



```
top - 16:40:46 up 3 min,  1 user,  load average: 0.01, 0.02, 0.00
Tasks:  86 total,   1 running,  85 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 99.8%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   2956912k total,   143492k used,  2813420k free,     5736k buffers
Swap:  1015800k total,        0k used,  1015800k free,    27260k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
    9 root      20   0     0    0    0 S  0.3  0.0   0:00.56 ksoftirqd/1
 1010 root      20   0 15016 1264  984 R  0.3  0.0   0:00.34 top
    1 root      20   0 19228 1496 1228 S  0.0  0.1   0:01.86 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.01 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.05 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.04 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:01.01 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
   11 root      20   0     0    0    0 S  0.0  0.0   0:00.04 events/0
   12 root      20   0     0    0    0 S  0.0  0.0   0:00.20 events/1
   13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cgroup
   14 root      20   0     0    0    0 S  0.0  0.0   0:00.00 khelper
   15 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns
   16 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr
   17 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm
   18 root      20   0     0    0    0 S  0.0  0.0   0:00.00 sync_supers
   19 root      20   0     0    0    0 S  0.0  0.0   0:00.00 bdi-default
   20 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0
   21 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/1
   22 root      20   0     0    0    0 S  0.0  0.0   0:00.15 kblockd/0
   23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kblockd/1
   24 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpid
   25 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
   26 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpi_hotplug
   27 root      20   0     0    0    0 S  0.0  0.0   0:00.01 ata/0
   28 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ata/1
   29 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ata_aux
```

Results - No IPS

After Attack

```
top - 16:42:36 up 5 min,  1 user,  load average: 16.62, 4.08, 1.35
Tasks: 126 total,   1 running, 125 sleeping,   0 stopped,   0 zombie
Cpu(s):  6.4%us, 14.5%sy,  0.0%ni,  0.0%id, 70.7%wa,  2.3%hi,  6.1%si,  0.0%st
Mem:   2956912k total,  2898520k used,    58392k free,       92k buffers
Swap:  1015800k total,   971996k used,    43804k free,     2804k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
   22 root      20   0     0    0    0 S  3.8  0.0   0:00.96 kblockd/0
 1044 daemon    20   0  113m  86m  140 D  3.1  3.0   0:00.66 httpd
 1042 daemon    20   0  118m  92m  140 D  2.8  3.2   0:00.61 httpd
    9 root      20   0     0    0    0 S  2.5  0.0   0:01.07 ksoftirqd/1
 1015 daemon    20   0  160m  55m   12 D  2.5  1.9   0:00.92 httpd
 1024 daemon    20   0  149m  86m  136 D  2.2  3.0   0:00.75 httpd
 1027 daemon    20   0  113m  76m  136 D  2.2  2.7   0:00.63 httpd
 1041 daemon    20   0  118m  81m  140 D  2.2  2.8   0:00.62 httpd
   12 root      20   0     0    0    0 S  2.0  0.0   0:00.45 events/1
 1014 daemon    20   0  160m  56m   12 D  2.0  2.0   0:00.93 httpd
 1019 daemon    20   0  140m  75m  136 D  2.0  2.6   0:00.75 httpd
 1036 daemon    20   0  114m  77m  136 D  2.0  2.7   0:00.62 httpd
    4 root      20   0     0    0    0 S  1.9  0.0   0:00.23 ksoftirqd/0
 1018 daemon    20   0  131m  67m  136 D  1.9  2.3   0:00.70 httpd
 1035 daemon    20   0  119m  81m  136 D  1.9  2.8   0:00.60 httpd
 1039 daemon    20   0  109m  74m  140 D  1.9  2.6   0:00.59 httpd
 1046 daemon    20   0 98.8m  71m  140 D  1.9  2.5   0:00.52 httpd
 1047 daemon    20   0  114m  87m  140 D  1.9  3.0   0:00.62 httpd
 1021 daemon    20   0  130m  66m  136 D  1.7  2.3   0:00.66 httpd
 1023 daemon    20   0  127m  66m  136 D  1.7  2.3   0:00.67 httpd
 1025 daemon    20   0  133m  71m  136 D  1.7  2.5   0:00.73 httpd
 1029 daemon    20   0  115m  78m  136 D  1.7  2.7   0:00.60 httpd
 1033 daemon    20   0  120m  83m  136 D  1.7  2.9   0:00.59 httpd
 1040 daemon    20   0  113m  78m  140 D  1.7  2.7   0:00.58 httpd
 1043 daemon    20   0 99964  71m  140 D  1.7  2.5   0:00.52 httpd
   38 root      20   0     0    0    0 D  1.6  0.0   0:00.47 kswapd0
 1016 daemon    20   0  160m  60m   12 D  1.6  2.1   0:00.86 httpd
 1017 daemon    20   0  160m  53m   12 D  1.6  1.8   0:00.84 httpd
 1028 daemon    20   0  106m  70m  136 D  1.6  2.5   0:00.58 httpd
 1031 daemon    20   0  103m  68m  136 D  1.6  2.4   0:00.57 httpd
```

Results - With IPS

When exploit is run, there is no noticeable increase in memory consumption or CPU load on the App Server.

Legitimate client requests, made by w3m on the Attacker's machine, are allowed through and their responses are sent back without error

Results - With IPS

Excerpt of log output

```
Dec  6 02:39:21 ipsrouter reverse_proxy[1313]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1314]: data from 7.0.0.2 was buffered
Dec  6 02:39:21 ipsrouter reverse_proxy[1314]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1315]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1316]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1317]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1318]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1319]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1320]: data from 7.0.0.2 was rejected
Dec  6 02:39:21 ipsrouter reverse_proxy[1321]: data from 7.0.0.2 was rejected
```

Conclusion

It works!

Conclusion

Improvements

- send warning email to sys admin
- decrease verbosity of output
- log all traffic from IP address of attacker
- add HTTPS support

Conclusion

Caveats to custom IPS

- should only be temporary and last resort
  - use official patch or workaround if available
- Custom IPS may have its own vulnerabilities
  - better to use tried and tested solution
  - ModSecurity

Conclusion

This project has expanded my knowledge of

- C programming in a Linux environment
- network protocols (TCP/IP, HTTP)
- system and network administration
- protecting against application layer remote exploits

Questions?

???