MANDIANT®
A FireEye® Company

REPORT

SYNful Knock
A CISCO IMPLANT

SEPTEMBER 2015

AUTHORS:
BILL HAU
TONY LEE
JOSH HOMAN

SECURITY
CONSULTING

CONTENTS

Executive Overview

Using the analogy of medieval castles, for the past 15 years we have been protecting the castle and its treasures with layered tiers of multiple large stone walls with restricted access into and out of the castle. All the while the defenders of the castle have been sitting high on the battlements trying to defend against the attacks they can see.

The fundamental assumption in this defensive strategy lies in the belief that we have dug the foundations to these large stone walls deep enough so we don't need to worry about what happens below ground. Any attack below the ground surface was deemed mostly theoretical in nature.

After the discovery of a Cisco router implant, dubbed SYNful Knock, this report will demonstrate that this supposed theoretical attack is now a reality; it will walk through the details of the compromise, impact, detection, and finally, its remediation.

We believe our findings represent the tip of the iceberg on this issue. Further research will need to be undertaken to assess the extent of the issue. In our analogy, the castle walls are akin to the traditional legacy security technologies available, such as firewalls, proxies, IDS, antivirus, etc. The foundation is akin to the building blocks of the Internet: the enterprise router.

We already know that the large stone walls above the ground are not protecting our governments / organizations. The publicly disclosed 2015 breaches at the Office of Personnel Management OPM (USA), Kaspersky Lab (Russia), Japan Pension Service (Japan) and Carphone Warehouse (UK) are perfect examples.

As no one is really monitoring below the castle walls, we hope to reinforce the need for governments and organizations to understand that the barbarians may have already dug under the gates and they are already inside the castle.

Building a fortress on a questionable foundation **does not guarantee security.**

UKRAINE
MEXICO
PHILIPPINES
INDIA

Mandiant's experience shows that this is not just an isolated issue, but a global one.

How can this happen?

Router implants, from any vendor in the enterprise space, have been largely believed to be theoretical in nature and especially in use. However, recent vendor advisories indicate that these have been seen in the wild. Mandiant can confirm the existence of at least 14 such router implants spread across four different countries: Ukraine, Philippines, Mexico, and India.

SYNful Knock is a stealthy modification of the router's firmware image that can be used to maintain persistence within a victim's network. It is customizable and modular in nature and thus can be updated once implanted. Even the presence of the backdoor can be difficult to detect as it uses non-standard packets as a form of pseudo-authentication.

The initial infection vector does not appear to leverage a zero-day vulnerability. It is believed that the credentials are either default or discovered by the attacker to install the backdoor. However, the router's position in the network makes it an ideal target for re-entry or further infection.

Finding backdoors within your network can be challenging. Finding a router implant? Even more so. This report not only dissects the implant, but also provides practical methods and tools for detecting a SYNful Knock compromise.

The impact of finding this implant on your network is severe and most likely indicates the presence of other footholds or compromised systems. This backdoor provides ample capability for the attacker to propagate and compromise other hosts and critical data using this as a very stealthy beachhead.

Malicious Program
Analysis of the SYNful Knock Cisco Implant

Summary

The implant consists of a modified Cisco IOS image that allows the attacker to load different functional modules from the anonymity of the Internet. The implant also provides unrestricted access using a secret backdoor password. Each of the modules is enabled via the HTTP protocol (not HTTPS), using a specifically crafted TCP packet sent to the router's interface. The packets utilize nonstandard sequence and corresponding acknowledgment numbers. The modules can manifest themselves as independent executable code or hooks within the router's IOS that provide functionality similar to the backdoor password. The backdoor password provides access to the router through the console and Telnet as well as privilege escalation via the enable command.

FIGURE 1: Module Updates

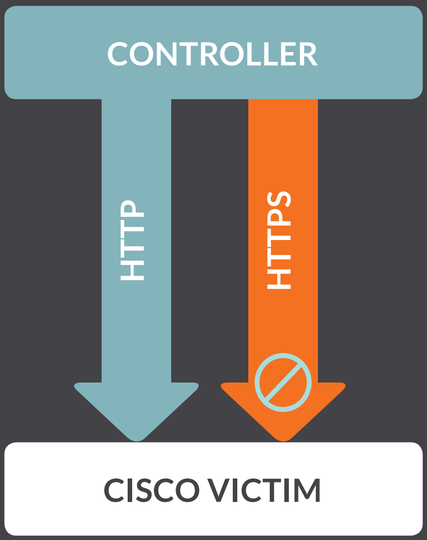
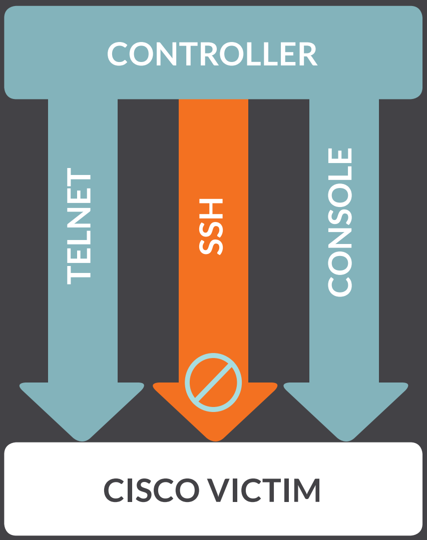


FIGURE 2: Backdoor Access



Known Affected Hardware

Cisco 1841 router	Cisco 2811 router	Cisco 3825 router

Note: Our initial identification revealed that other models are likely affected based on the similarity in core functionality and IOS code base.

Persistence

The implant resides within a modified Cisco IOS image and, when loaded, maintains its persistence in the environment, even after a system reboot. However, any further modules loaded by the attacker will only exist in the router's volatile memory and will not be available for use after reboot. From a forensic standpoint, if the modules are loaded in volatile memory, one can analyze them by obtaining a core dump of the router image .[1]

Detection Methodology

Host-based indicators are useful for organizations that can issue commands and receive the responses. This will be feasible for a small amount of routers located in easily accessible areas of the network.

Network-based indicators will assist organizations that are more dispersed or those that lack the ability to easily execute local commands and receive the responses.

Ultimately a combination of host and network-based indicators will most likely be used to determine the health of the underlying network.

Host-Based Indicators

If command-line access is possible, the following techniques can be used to detect the implant:

In addition to the command in Table 1, other detection techniques are contained within Cisco's IOS Integrity Assurance document:

http://www.cisco.com/web/about/security/intelligence/integrity-assurance.html

In the case of this implant, the size of the implanted IOS binary is the same size as the legitimate image. Thus if comparing file size, it will appear to be unmodified. Hashing the image and comparing the result to the hash from Cisco is one of the best methods to detect a modified binary; however, this will only work with an image on disk and not one that is loaded into memory.

Network-Based Indicators

Mandiant provides both active and passive network detection capabilities. A detailed description of both methods is provided below (see Network Detection).

TABLE 1: Host-based indicator command and expected output

COMMAND	EXPECTED OUTPUT
`"show platform	include RO, Valid"`	Implanted router may produce no results

[1] The following blog entry shows how to produce a Cisco IOS core dump: http://blogs.cisco.com/security/offline-analysis-of-ios-image-integrity

DETAILS

Modifications to the IOS binary can be broken down into the following four functions:

1

Modify the translation look aside buffer (TLB) Read/ Write attributes

2

Modify a legitimate IOS function to call and initialize the malware

3

Overwrite legitimate protocol handling functions with malicious code

4

Overwrite strings referenced by legitimate functions with strings used by the malware

1. TLB Read/Write Attributes

The malware forces all TLB Read and Write attributes to be Read-Write (RW). We believe this change is made to support the hooking of IOS functions by loaded modules. If the TLB attributes are not set to RW, modifications to the cached pages may not be propagated to the original page in memory.

This is accomplished with two single-byte modifications made to the IOS function suspected to be responsible for configuring the TLB. The unmodified function sets the first two bits of a register to 1, and the modified function sets the first three bits to 1. Mandiant believes that the third bit controls the Write permissions on the TLB entry. Figure 3 shows the modified instructions.

FIGURE 3: Modification to TLB attributes

```
Original:
.text:XXXXXXXX 36 D2 00 03            ori    $s2, $s6, 3
.text:XXXXXXXX 36 91 00 03            ori    $s1, $s4, 3

Modified
.text:XXXXXXXX 36 D2 00 07            ori    $s2, $s6, 7
.text:XXXXXXXX 36 91 00 07            ori    $s1, $s4, 7
```

This brings us to one of the host-based indicators discussed above. The TLB attributes can be examined using the enable mode command "show platform". The TLB output of an unmodified IOS image is shown below in Figure 4.

FIGURE 4: TLB entries for a legitimate IOS image

```
16M  0xX0000000:0xXXFFFFFF   0xX0000000:0xXXFFFFFF   CacheMode=2, RW, Valid
16M  0xXX000000:0xXXFFFFFF   0xXX000000:0xXXFFFFFF   CacheMode=2, RW, Valid
16M  0xX0000000:0xXXFFFFFF   0x00000000:0x0XFFFFFF   CacheMode=3, RO, Valid
4M   0xXX000000:0xXXXFFFFF   0x0X000000:0x0XXFFFFF   CacheMode=3, RO, Valid
256K 0xXXX00000:0xXXXXFFFF   0x0XX00000:0x0XXXFFFF   CacheMode=3, RO, Valid
256K 0xXXXX0000:0xXXXFFFFF   0x0XXX0000:0x0XXFFFFF   CacheMode=3, RO, Valid
256K 0xXXX00000:0xXXXXFFFF   0x0XX00000:0x0XXXFFFF   CacheMode=3, RW, Valid
256K 0xXXXX0000:0xXXXFFFFF   0x0XXX0000:0x0XXFFFFF   CacheMode=3, RW, Valid
```

If the router has been implanted with a modified IOS image, the RW attributes should be:

FIGURE 5: TLB entries for a modified IOS image

```
16M   0xX0000000:0xXXFFFFFF   0xX0000000:0xXXFFFFFF   CacheMode=2, RW, Valid
16M   0xXX000000:0xXXFFFFFF   0xXX000000:0xXXFFFFFF   CacheMode=2, RW, Valid
16M   0xX0000000:0xXXFFFFFF   0x00000000:0x0XFFFFFF   CacheMode=3, RW, Valid
4M    0xXX000000:0xXXXFFFFF   0x0X000000:0x0XXFFFFF   CacheMode=3, RW, Valid
256K  0xXXX00000:0xXXXXFFFF   0x0XX00000:0x0XXXFFFF   CacheMode=3, RW, Valid
256K  0xXXXX0000:0xXXXFFFFF   0x0XXX0000:0x0XXFFFFF   CacheMode=3, RW, Valid
256K  0xXXX00000:0xXXXXFFFF   0x0XX00000:0x0XXXFFFF   CacheMode=3, RW, Valid
256K  0xXXXX0000:0xXXXFFFFF   0x0XXX0000:0x0XXFFFFF   CacheMode=3, RW, Valid
```

Depending on router hardware, certain ranges of memory addresses are typically read-only executable code sections. The simplest way to determine if the router has been modified is to use the "show platform | include RO, Valid" command. The IOS image may have been tampered with to allow the modification of executable code if no results are displayed.

2. Initialize the Malware

To execute the malware during IOS image loading, Mandiant believes a function associated with process scheduling was modified. This was chosen because the modified function is called early on during the IOS boot sequence, and is always called, as long as the IOS boots correctly. The target address of a function call is modified to point to the malware hook processing function. Our research has shown that the malware is initialized after the hook processing function checks whether the calling function is valid in the modified IOS. Now that the malware is up and running, it executes the original IOS function so no one is the wiser.

Mandiant believes the modified function is linked with the process scheduling task, which enters an infinite loop once called. In addition, several of the sub functions reference strings associated with process scheduling, such as "Threshold: %s CPU Utilization(Total/Intr):...".

3. Malware Executable Code Placement

To prevent the size of the image from changing, the malware overwrites several legitimate IOS functions with its own executable code. The attackers will examine the current functionality of the router and determine functions that can be overwritten without causing issues on the router. Thus, the overwritten functions will vary upon deployment.

4. Malware Strings and Configuration

Keeping with the theme mentioned above, since the image size cannot change, the implant also overwrote some reporting strings with its own configuration. This is another indicator of compromise that can be used for detection purposes. The legitimate strings that are overwritten are shown in Figure 6.

FIGURE 6: Strings associated with a valid function overwritten by the malware

```
XXXXXXXX  65 63 20 00 2C 20 43 6F 6E 66 69 67 75 72 65 64  ec ., Configured
XXXXXXXX  20 49 6E 74 65 72 76 61 6C 20 25 64 20 73 65 63   Interval %d sec
XXXXXXXX  00 00 00 00 0A 4E 65 78 74 20 75 70 64 61 74 65  .....Next update
XXXXXXXX  20 64 75 65 20 69 6E 20 00 00 00 00 0A 43 75 72   due in .....Cur
XXXXXXXX  72 65 6E 74 20 74 69 6D 65 20 25 54 61 00 00 00  rent time %Ta...
XXXXXXXX  0A 49 6E 64 65 78 20 25 64 20 54 69 6D 65 73 74  .Index %d Timest
XXXXXXXX  61 6D 70 20 25 54 61 00 0A 0A 46 61 69 6C 75 72  amp %Ta...Failur
XXXXXXXX  65 20 48 65 61 64 20 25 64 2C 20 4C 61 73 74 20  e Head %d, Last
XXXXXXXX  25 64 20 4C 53 41 20 67 72 6F 75 70 20 66 61 69  %d LSA group fai
XXXXXXXX  6C 75 72 65 20 6C 6F 67 67 65 64 00 0A 54 69 6D  lure logged..Tim
XXXXXXXX  65 20 20 20 20 20 20 20 20 20 44 65 6C 61 79 20  e         Delay
XXXXXXXX  20 20 20 20 20 20 20 4A 2D 44 65 6C 61 79 20 20         J-Delay
XXXXXXXX  20 20 20 20 53 74 61 72 74 20 54 69 6D 65 73 74      Start Timest
XXXXXXXX  61 6D 70 20 20 20 45 6E 64 20 54 69 6D 65 73 74  amp   End Timest
XXXXXXXX  61 6D 70 00 0A 25 31 33 54 61 25 31 33 54 61 25  amp..%13Ta%13Ta%
XXXXXXXX  31 33 54 61 25 31 38 54 61 25 32 30 54 61 00 00  13Ta%18Ta%20Ta..
```

The contents shown in Figure 6 were replaced with the contents shown below in Figure 7. Clearly visible are the malware's strings (included in the HTTP header used in Command and Control (CnC), along with the default password, which we have intentionally blanked. This will provide potential victims time to search their own networks for compromise and remediate the issue. Feel free to contact us via email at synfulknock [at] fireeye.com and we can provide the password if you suspect your system is compromised.

FIGURE 7: Malware strings

```
XXXXXXXX  00 00 00 00 00 00 00 00 00 00 00 00 48 54 54 50  ............HTTP
XXXXXXXX  2F 31 2E 31 20 32 30 30 20 4F 4B 0D 0A 53 65 72  /1.1 200 OK..Ser
XXXXXXXX  76 65 72 3A 20 41 70 61 63 68 65 2F 32 2E 32 2E  ver: Apache/2.2.
XXXXXXXX  31 37 20 28 55 62 75 6E 74 75 29 0D 0A 58 2D 50  17 (Ubuntu)..X-P
XXXXXXXX  6F 77 65 72 65 64 2D 42 79 3A 20 50 48 50 2F 35  owered-By: PHP/5
XXXXXXXX  2E 33 2E 35 2D 31 75 62 75 6E 74 75 37 2E 37 0D  .3.5-1ubuntu7.7.
XXXXXXXX  0A 4B 65 65 70 2D 41 6C 69 76 65 3A 20 74 69 6D  .Keep-Alive: tim
XXXXXXXX  65 6F 75 74 3D 31 35 2C 20 6D 61 78 3D 31 30 30  eout=15, max=100
XXXXXXXX  0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B 65  ..Connection: Ke
XXXXXXXX  65 70 2D 41 6C 69 76 65 0D 0A 43 6F 6E 74 65 6E  ep-Alive..Conten
XXXXXXXX  74 2D 54 79 70 65 3A 20 74 65 78 74 2F 68 74 6D  t-Type: text/htm
XXXXXXXX  6C 0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C 62 6F 64 79  l....<html><body
XXXXXXXX  3E 3C 64 69 76 3E 00 00 3C 2F 64 69 76 3E 3C 2F  ><div>..</div></
XXXXXXXX  62 6F 64 79 3E 3C 2F 68 74 6D 6C 3E 00 00 00 00  body></html>....
XXXXXXXX  34 30 34 0A 00 00 00 00 00 00 00 00 00 00 00 00  404.............
XXXXXXXX  ** ** ** ** ** ** ** ** ** ** ** ** ** 00 00 00  ***blanked***...
```

Again we arrive at another host-based indicator that can potentially be used to identify the presence of the implant; however the location of the configuration strings must first be discovered and may vary depending on deployment.

A modified IOS image will produce a very different and suspicious result when running what would seem to be an ordinary IOS command. Suspicious example output is shown below:

Depending on the implant, backdoor headers may be displayed after running an legitimate IOS command.

```
<html><body><div>.3.5-1ubuntu7.7
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

Backdoor Password

The attacker can utilize the secret backdoor password in three different authentication scenarios. The implant will first check whether the user input is the backdoor password. If so, access is granted. Otherwise, the implanted code will pass the credentials on for verification of potentially valid credentials. This raises the least amount of suspicion.

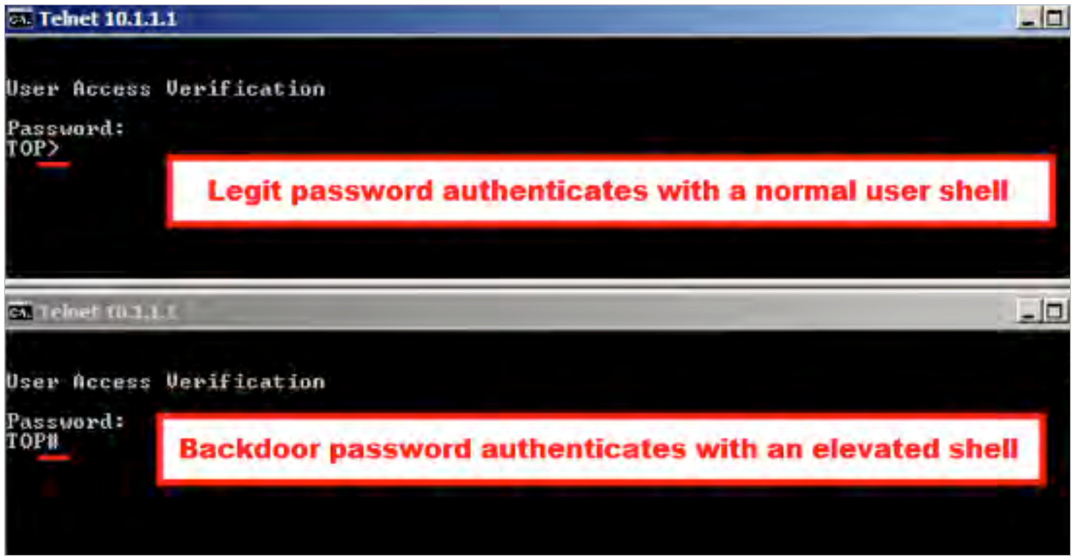
The following three instances were verified to enable access using the backdoor password:

TABLE 2: Authentication functions in which the secret backdoor password can be used

METHOD	PROMPT	RESULTS
Console	"User Access Verification"	Access and elevated session
Telnet	Username is the backdoor password	Access and elevated session
Elevation (enable)	Enable password	Elevated session

However, this research has shown that SSH or HTTPS sessions do not provide access for the backdoor password. This could be a configuration issue and may vary based on compromise.

FIGURE 8: Subtle difference between authenticating using a legitimate password and the backdoor password



```
User Access Verification

Password:
TOP>
```

Legit password authenticates with a normal user shell
Telnet 10.1.1.1

```
User Access Verification

Password:
TOP#
```

Backdoor password authenticates with an elevated shell

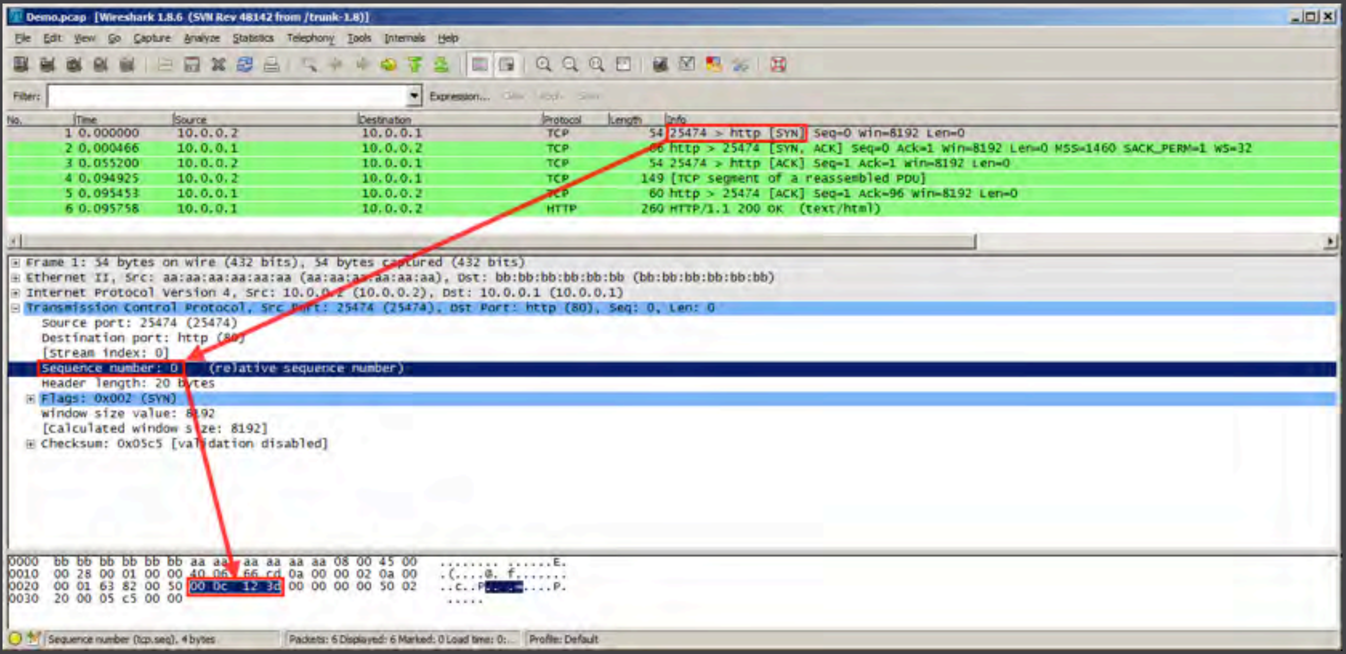
Network Command and Control

The Command and Control (CnC) portion of the implant is modular and allows additional functionality to be loaded into the IOS. The CnC functionality is stealthy because it requires a series of TCP trigger packets that the malware monitors for specific TCP header values and content. Even if filters are enabled on the router, the TCP trigger is processed by the malware. The malware will respond to trigger packets sent from three different addresses: the router interface, the broadcast IP, and the network address (the first IP in a subnet).

1

To initiate the process, a uniquely crafted TCP SYN packet is sent to port 80 of the implanted router. It is important to note that the difference between the sequence and acknowledgment numbers must be set to 0xC123D. Also the ACK number doesn't need to be zero.

FIGURE 9: TCP SYN with sequence and acknowledgement difference of 0xC123D


Demo.pcap [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...

```
No.  Time        Source     Destination  Protocol Length Info
  1 0.000000    10.0.0.2   10.0.0.1     TCP        54 25474 > http [SYN] Seq=0 Win=8192 Len=0
  2 0.000466    10.0.0.1   10.0.0.2     TCP        66 http > 25474 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 SACK_PERM=1 WS=32
  3 0.055200    10.0.0.2   10.0.0.1     TCP        54 25474 > http [ACK] Seq=1 Ack=1 Win=8192 Len=0
  4 0.094925    10.0.0.2   10.0.0.1     TCP       149 [TCP segment of a reassembled PDU]
  5 0.095453    10.0.0.1   10.0.0.2     TCP        60 http > 25474 [ACK] Seq=1 Ack=96 Win=8192 Len=0
  6 0.095758    10.0.0.1   10.0.0.2     HTTP      260 HTTP/1.1 200 OK  (text/html)

⊞ Frame 1: 54 bytes on wire (432 bits), 54 bytes captured (432 bits)
⊞ Ethernet II, Src: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa), Dst: bb:bb:bb:bb:bb:bb (bb:bb:bb:bb:bb:bb)
⊞ Internet Protocol Version 4, Src: 10.0.0.2 (10.0.0.2), Dst: 10.0.0.1 (10.0.0.1)
⊟ Transmission Control Protocol, Src Port: 25474 (25474), Dst Port: http (80), Seq: 0, Len: 0
    Source port: 25474 (25474)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Header length: 20 bytes
  ⊞ Flags: 0x002 (SYN)
    Window size value: 8192
    [Calculated window size: 8192]
  ⊞ Checksum: 0x05c5 [validation disabled]

0000  bb bb bb bb bb bb aa aa  aa aa aa aa 08 00 45 00   ........ ......E.
0010  00 28 00 01 00 00 40 06  66 cd 0a 00 00 02 0a 00   .(....@. f.......
0020  00 01 63 82 00 50 00 0c  12 3d 00 00 00 00 50 02   ..c..P.. .=....P.
0030  20 00 05 c5 00 00                                   ......

Sequence number (tcp.seq), 4 bytes | Packets: 6 Displayed: 6 Marked: 0 Load time: 0:... | Profile: Default
```

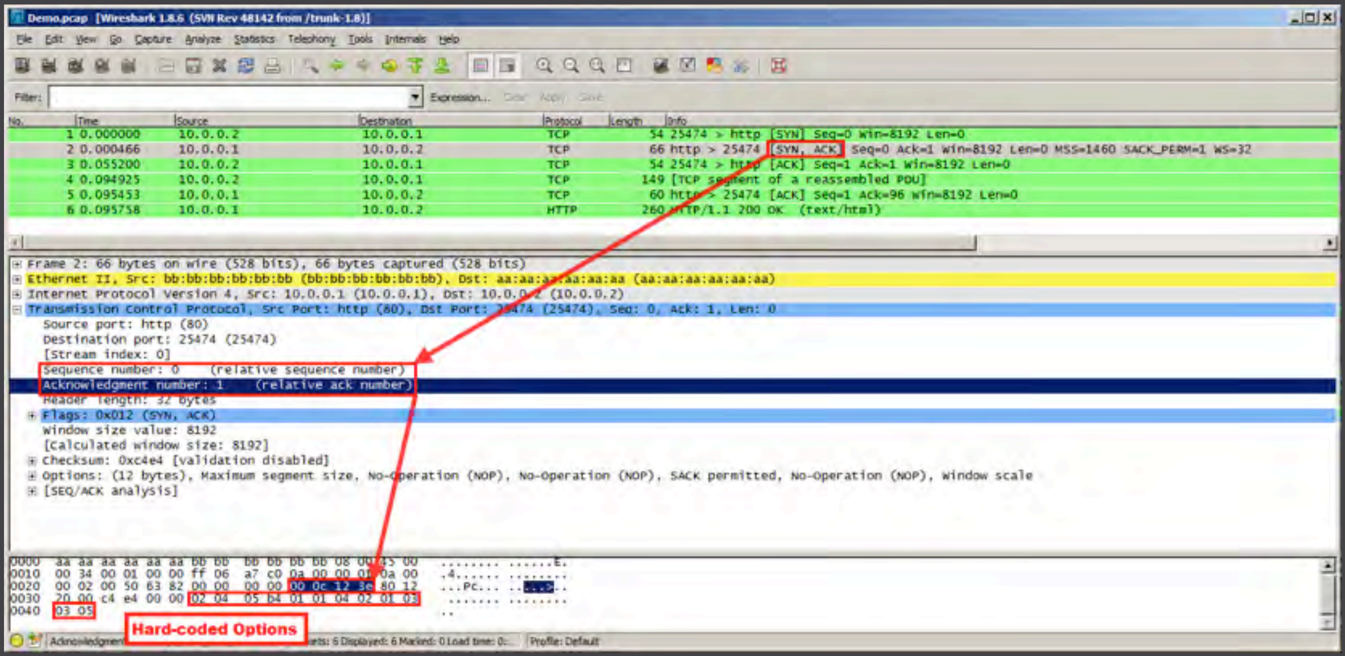
2

As typical with a 3-way handshake, the malware responds with a TCP SYN-ACK message acknowledging the first SYN message. However, the following conditions will be present:

- The differential between the acknowledgment and sequence numbers is now 0xC123E
- The following hard-coded TCP options are set: "`02 04 05 b4 01 01 04 02 01 03 03 05`"
- The urgent pointer is also set to 0x0001 but the urgent flag is not set
- The malware also copies the acknowledgment number from the SYN packet for the sequence number. A typical server usually generates a random sequence number, thus this is not a standard TCP handshake.

Unique conditions cause anomalies that allow Mandiant to write network detection signatures and tools.

FIGURE 10: TCP SYN-ACK with sequence and acknowledgement offset of 0xC123E


Demo.pcap [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: Expression... Clear Apply Save
Hard-coded Options
Acknowledgment
Packets: 6 Displayed: 6 Marked: 0 Load time: 0:...
Profile: Default

```
No.  Time        Source     Destination  Protocol  Length  Info
  1 0.000000    10.0.0.2   10.0.0.1     TCP        54 25474 > http [SYN] Seq=0 Win=8192 Len=0
  2 0.000466    10.0.0.1   10.0.0.2     TCP        66 http > 25474 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 SACK_PERM=1 WS=32
  3 0.055200    10.0.0.2   10.0.0.1     TCP        54 25474 > http [ACK] Seq=1 Ack=1 Win=8192 Len=0
  4 0.094925    10.0.0.2   10.0.0.1     TCP       149 [TCP segment of a reassembled PDU]
  5 0.095453    10.0.0.1   10.0.0.2     TCP        60 http > 25474 [ACK] Seq=1 Ack=96 Win=8192 Len=0
  6 0.095758    10.0.0.1   10.0.0.2     HTTP      260 HTTP/1.1 200 OK  (text/html)

⊞ Frame 2: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
⊞ Ethernet II, Src: bb:bb:bb:bb:bb:bb (bb:bb:bb:bb:bb:bb), Dst: aa:aa:aa:aa:aa:aa (aa:aa:aa:aa:aa:aa)
⊞ Internet Protocol Version 4, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.2 (10.0.0.2)
⊟ Transmission Control Protocol, Src Port: http (80), Dst Port: 25474 (25474), Seq: 0, Ack: 1, Len: 0
    Source port: http (80)
    Destination port: 25474 (25474)
    [Stream index: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 1    (relative ack number)
    Header length: 32 bytes
  ⊞ Flags: 0x012 (SYN, ACK)
    Window size value: 8192
    [Calculated window size: 8192]
  ⊞ Checksum: 0xc4e4 [validation disabled]
  ⊞ Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale
  ⊞ [SEQ/ACK analysis]

0000  aa aa aa aa aa aa bb bb  bb bb bb bb 08 00 45 00   ........ ......E.
0010  00 34 00 01 00 00 ff 06  a7 c0 0a 00 00 01 0a 00   .4...... ........
0020  00 02 00 50 63 82 00 00  00 00 00 0c 12 3e 80 12   ...Pc... .....>..
0030  20 00 c4 e4 00 00 02 04  05 b4 01 01 04 02 01 03    ....... ........
0040  03 05                                              ..
```

3

After the final ACK to complete the 3-way handshake, the controller then sends the following TCP message:

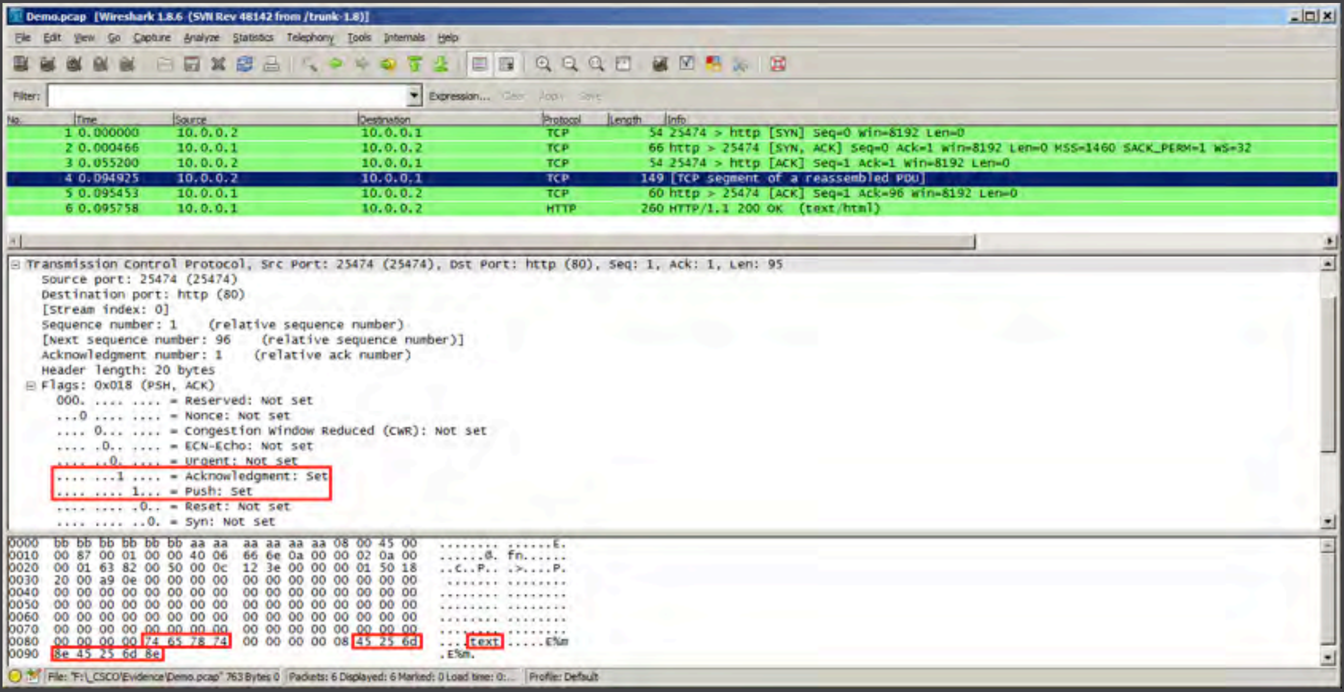
- The PUSH and ACK flags are set
- From the start of the TCP header, at offset 0x62, the string “text” is written
- The command shown below is at offset 0x67 from the TCP header

The command is in the following format:

```
[4 byte Command Length][CMD Data][4 byte checksum]
```

The [CMD Data] is XOR encoded with a static key. There is a checksum algorithm, which is a four-byte XOR of the decoded [CMD Data].

FIGURE 11: Controller command packet



Demo.pcap [Wireshark 1.8.6 (SVN Rev 48142 from /trunk-1.8)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time        Source     Destination  Protocol Length Info
   1 0.000000    10.0.0.2   10.0.0.1     TCP        54 25474 > http [SYN] Seq=0 Win=8192 Len=0
   2 0.000466    10.0.0.1   10.0.0.2     TCP        66 http > 25474 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 SACK_PERM=1 WS=32
   3 0.055200    10.0.0.2   10.0.0.1     TCP        54 25474 > http [ACK] Seq=1 Ack=1 Win=8192 Len=0
   4 0.094925    10.0.0.2   10.0.0.1     TCP       149 [TCP segment of a reassembled PDU]
   5 0.095453    10.0.0.1   10.0.0.2     TCP        60 http > 25474 [ACK] Seq=1 Ack=96 Win=8192 Len=0
   6 0.095758    10.0.0.1   10.0.0.2     HTTP      260 HTTP/1.1 200 OK  (text/html)
```

```
⊟ Transmission Control Protocol, Src Port: 25474 (25474), Dst Port: http (80), Seq: 1, Ack: 1, Len: 95
    Source port: 25474 (25474)
    Destination port: http (80)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 96    (relative sequence number)]
    Acknowledgment number: 1    (relative ack number)
    Header length: 20 bytes
  ⊟ Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
```

```
0000  bb bb bb bb bb bb aa aa  aa aa aa aa 08 00 45 00   ........ ......E.
0010  00 87 00 01 00 00 40 06  66 6e 0a 00 00 02 0a 00   ......@. fn......
0020  00 01 63 82 00 50 00 0c  12 3e 00 00 00 01 50 18   ..c..P.. .>....P.
0030  20 00 a9 0e 00 00 00 00  00 00 00 00 00 00 00 00    ....... ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0080  00 00 00 00 74 65 78 74  00 00 00 00 08 45 25 6d   ....text .....E%m
0090  8e 45 25 6d 8e                                    .E%m.
```

File: "F:_CSCO\Evidence\Demo.pcap" 763 Bytes 0 | Packets: 6 Displayed: 6 Marked: 0 Load time: 0:... | Profile: Default

4

The malware response is encapsulated in the following static HTTP/HTML server response.

```
HTTP/1.1 200 OK
Server: Apache/2.2.17 (Ubuntu)
X-Powered-By: PHP/5.3.5-1ubuntu7.7
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
<html><body><div>[Response]</div></body></html>
```

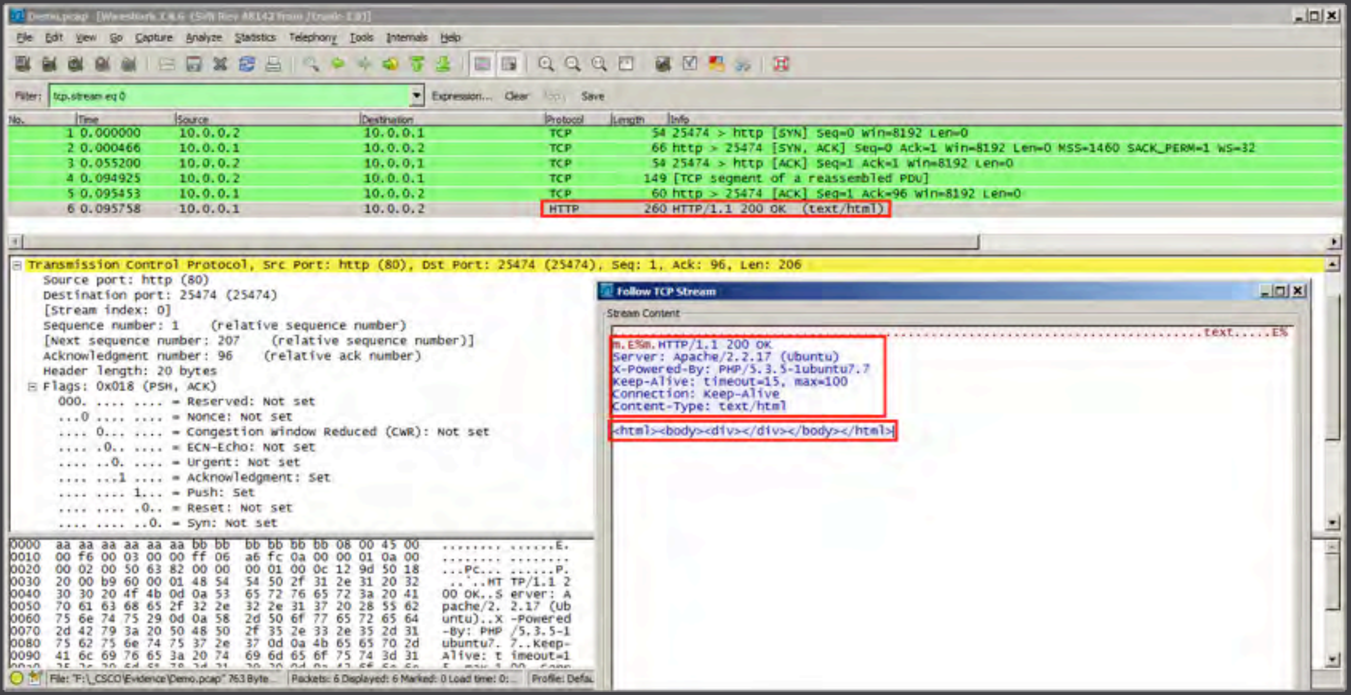


FIGURE 12: Victim response

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter: tcp.stream eq 0
Expression... Clear Apply Save

```
No.   Time        Source      Destination  Protocol  Length  Info
    1 0.000000    10.0.0.2    10.0.0.1     TCP           54 25474 > http [SYN] Seq=0 Win=8192 Len=0
    2 0.000466    10.0.0.1    10.0.0.2     TCP           66 http > 25474 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 SACK_PERM=1 WS=32
    3 0.055200    10.0.0.2    10.0.0.1     TCP           54 25474 > http [ACK] Seq=1 Ack=1 Win=8192 Len=0
    4 0.094925    10.0.0.2    10.0.0.1     TCP          149 [TCP segment of a reassembled PDU]
    5 0.095453    10.0.0.1    10.0.0.2     TCP           60 http > 25474 [ACK] Seq=1 Ack=96 Win=8192 Len=0
    6 0.095758    10.0.0.1    10.0.0.2     HTTP         260 HTTP/1.1 200 OK  (text/html)
```

```
⊟ Transmission Control Protocol, Src Port: http (80), Dst Port: 25474 (25474), Seq: 1, Ack: 96, Len: 206
    Source port: http (80)
    Destination port: 25474 (25474)
    [Stream index: 0]
    Sequence number: 1    (relative sequence number)
    [Next sequence number: 207    (relative sequence number)]
    Acknowledgment number: 96    (relative ack number)
    Header length: 20 bytes
  ⊟ Flags: 0x018 (PSH, ACK)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...1 .... = Acknowledgment: Set
      .... .... 1... = Push: Set
      .... .... .0.. = Reset: Not set
      .... .... ..0. = Syn: Not set
```

```
0000  aa aa aa aa aa aa bb bb  bb bb bb bb 08 00 45 00   ........ ......E.
0010  00 f6 00 03 00 00 ff 06  a6 fc 0a 00 00 01 0a 00   ........ ........
0020  00 02 00 50 63 82 00 00  00 01 00 0c 12 9d 50 18   ...Pc... ......P.
0030  20 00 b9 60 00 01 48 54  54 50 2f 31 2e 31 20 32    ..`..HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 53  65 72 76 65 72 3a 20 41   00 OK..S erver: A
0050  70 61 63 68 65 2f 32 2e  32 2e 31 37 20 28 55 62   pache/2. 2.17 (Ub
0060  75 6e 74 75 29 0d 0a 58  2d 50 6f 77 65 72 65 64   untu)..X -Powered
0070  2d 42 79 3a 20 50 48 50  2f 35 2e 33 2e 35 2d 31   -By: PHP /5.3.5-1
0080  75 62 75 6e 74 75 37 2e  37 0d 0a 4b 65 65 70 2d   ubuntu7. 7..Keep-
0090  41 6c 69 76 65 3a 20 74  69 6d 65 6f 75 74 3d 31   Alive: t imeout=1
```

File: "F:_CSCO\Evidence\Demo.pcap" 763 Byte...
Packets: 6 Displayed: 6 Marked: 0 Load time: 0:...
Profile: Defau
Follow TCP Stream
Stream Content

```
.................................................................text.....E%
m.E%m.HTTP/1.1 200 OK
Server: Apache/2.2.17 (Ubuntu)
X-Powered-By: PHP/5.3.5-1ubuntu7.7
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html

<html><body><div></div></body></html>
```

Supported Commands

We mentioned previously that this implant is modular. The five commands shown in the table below are used for loading additional modules and functionality on the victim's router. A total of 100 additional modules can be loaded; however, these modules are memory resident and will not persist after a reboot or reload.

Command messages set the first WORD (4-byte big-endian) to zero. The second WORD identifies the message type (values zero through four). All message types will start with the following eight bytes:

```
00 00 00 00 00 00 00 [00 – 04] [Optional Arguments]
```

TABLE 3: Supported Commands

ID	DESCRIPTION
0	**List loaded modules and their current state.** The response contains a word representing the ID number followed by a word representing the state for each loaded module. 00 - Memory is allocated 01 - Module is loaded into memory 02 - Module is activated For example, if the malware responds with this message: `00 00 00 03 00 00 02` Then, the message would indicate module `03` is in the activated state (`02`).

1	**Allocate space for an additional module to be loaded.** The command provides the module size for two required buffers. The malware allocates the memory for two buffers and returns the addresses in the response. The first buffer is the executable code, and we suspect that the second buffer is for configuration and storage. The syntax for this message follows this format:
	
	`[WORD ID][WORD first buffer length][WORD second buffer length]`
	
	An example command that tells the malware to allocate 0x0C bytes for the first buffer and 0x90 bytes for the second buffer of module ID 0x02:
	
	`00 00 00 02 00 00 00 0C 00 00 00 90`
	
	An example response from the server shows the first buffer is at memory address 0x66012C4C and the second is at 0x650DCD20:
	
	`66 01 2C 4C 65 0D CD 20`
	
	After executing this command, the module state is set to zero.

TABLE 3: Supported Commands continued

ID	DESCRIPTION

2	**Populate the memory allocated for the module.** This command is used to populate the executable code and suspected configuration data.
	`[0x80 Bytes hook data][WORD first buffer length][WORD second buffer length]`
	`[First buffer...][Second buffer...]`
	Similar to how the default password hook functions, the hook data buffer is used to inject additional hooks into the IOS. The hook buffer provides addresses within the IOS where hooks should be installed, and the code that should be run when the hooks are executed.
	After executing this command, the module state is set to one.

3	**Activate a loaded module.** The malware parses the hook data buffer and creates the necessary hooks within the OS to execute a module. The only argument is a WORD representing the module ID. After executing this command, the module state is set to two.
4	**Remove a module.** The memory allocated for the module is released and the state is set to zero. The module will no longer show up in the active modules command.

If the first WORD of a message is not zero, the code associated with the module ID of the first WORD is executed. This enables the execution of code that is not hooked into an IOS function.

Network Detection

Both active and passive network detection can be deployed to detect and prevent a SYNful Knock compromise. Passive detection can be incorporated into network defense sensors while active techniques can be used to hunt for the backdoor.

Passive Network Detection

There are a number of approaches for passive network detection. Our network detection signatures focus on four parts of a CnC session: the SYN, SYN-ACK, malware response messages and controller commands. An IDS must be able to monitor the external interface of the router to effectively detect this backdoor from the network.

1

SYN:

The first signature (included in Appendix A) detects SYN packets with the necessary delta between the TCP sequence and acknowledgment numbers. To reduce the chance of false positives, the signature assumes that the acknowledgement field is not set to zero. This signature detects probing for the malware, and does not necessarily indicate that the destination is compromised.

2

SYN/ACK:

The second signature (included in Appendix B) validates the delta between TCP sequence and acknowledgement numbers and the TCP options to detect the SYN ACK response from the malware. This signature does not assume that the acknowledgement in the SYN packet is not zero.

3

Malware response message:

The signature shown below detects the HTTP server response when a command is issued. The advantage of the signature below is that it is a standard snort signature; however, it does not have the capability to validate the delta between the TCP sequence and acknowledgment numbers.

```
alert tcp any any -> any any (\
msg: “SYNful Knock Cisco Implant HTTP Header”;\
flow: from_server;\
content: “HTTP/1.1 200 OK|0d 0a|Server: Apache/2.2.17 (Ubuntu)|0d 0a|X-Powered-By: PHP/5.3.5-
1ubuntu7.7|0d 0a|Keep-Alive: timeout=15, max=100|0d 0a|Connection: Keep-Alive|0d 0a|Content-Type:
text/html|0d 0a 0d 0a|<html><body><div>”; offset:0;\
flags:PA;\
sid:201504232;\
```

4

Controller commands:

The following signature detects a command issued from the controller. It uses the “text” string at the location expected by the malware and message size less than 256 bytes. The signature also assumes no TCP header options are present. If TCP header options are included, this signature may need to be converted to a compiled signature or multiple variants created to deal with each length.

```
alert tcp any any -> any any (\
msg: “SYNful Knock Cisco Implant HTTP Request”;\
flow: to_server;\
content: “text”; offset:78; depth:4;\
content: “|00 00 00|”; offset: 83; depth: 3;\
content: “|45 25 6d|”; offset: 87; depth: 3;\
sid:201504233;\
)
```

Active Network Detection

Nmap Scripting Engine (NSE)

Mandiant authored an NSE script in LUA to actively scan for the presence of this Cisco implant.

REQUIREMENTS

- Nmap v6.47 or higher (also tested with v6.49)
- Modified nselib

MODIFIED NSE LIBRARY

The NSE packet library does not allow the user to modify ack values; thus, Mandiant modified the library to allow for this capability. The diff is shown below.

```
/usr/share/nmap/nselib# diff packet.lua packet2.lua
1013a1014,1021
>
> --- Set the TCP acknowledgment field.
> -- @param new_ack Acknowledgment.
> function Packet:tcp_set_ack(new_ack)
>   self:set_u32(self.tcp_offset + 8, new_ack)
>   self.tcp_ack = new_ack
> end
```

ESTIMATED WORST-CASE SPEED (THIS FACTORS IN HIGH UNUSED IP SPACE)

Mandiant first ran this scan using Nmap's default scan speed of -T3:

```
nmap -sS -PN -n -T3 -p 80 --script="SYNfulKnock" 10.1.1.1/24
```
Class C - 256 IP addresses (4 hosts up) - scanned in 2.29 seconds

Mandiant then ran this scan using Nmap's scan speed of -T4:

```
nmap -sS -PN -n -T4 -p 80 --script="SYNfulKnock" 10.1.1.1/24
```
Class C - 256 IP addresses (4 hosts up) - scanned in 2.28 seconds

```
nmap -sS -PN -n -T4 -p 80 --script="SYNfulKnock" 10.1.1.1/16
```
Class B - 65536 IP addresses (4 hosts up) - scanned in 2557.50 seconds (42 min)

```
nmap -sS -PN -n -T4 -p 80 --script="SYNfulKnock" 10.1.1.1/8
```
Class A - 16,777,216 IP addresses - Estimated scan time = 10,752 minutes (179 hours) = 7 days

FLAG EXPLANATION:

```
-sS = SYN scan

-PN = Don't perform host discovery

-n = Don't perform name resolution

-T4 = Throttle to speed 4

-p = port number

--script = script to execute

optional:  --scriptargs="reportclean=1"  Shows the seq and ack for clean
devices too
```

Python Detection Script

Mandiant also authored a Python script to actively scan for this Cisco implant's presence. This script sends a crafted TCP SYN packet and analyzes the SYN/ACK response for indications of the implant. The script relies on the Scapy packet manipulation library (http://www.secdev.org/projects/scapy/) for processing, sending, and receiving packets. The scanning process uses several scan threads and a single thread for collecting the responses. This script is about 30 times slower than leveraging the nmap LUA script above; however, it is useful for small scans and verifying the faster scan.

REQUIREMENTS

- Python

SPEED

Class C - 256 IP addresses (4 hosts up) - 59.26 seconds
Class B - Terminated the script early due to time

COMMAND LINE

```
python ./SYNfulKnock_scanner.py -D 10.1.1.1/10.1.1.2
```

FLAG EXPLANATION:

```
-d = Target to be scanned (IP, IP/CIDR, First IP/Last IP)
```

OUTPUT

```
python ./SYNfulKnock_scanner.py -D 10.1.1.1/10.1.1.2
2015-07-14 12:59:02,760 190 INFO    Sniffer daemon started
2015-07-14 12:59:02,761 218 INFO    Sending 2 syn packets with 10 threads
2015-07-14 12:59:03,188 110 INFO    10.1.1.1:80 - Found implant seq: 667f6e09 ack: 66735bcd
2015-07-14 12:59:03,190 225 INFO    Waiting to complete send
2015-07-14 12:59:03,190 227 INFO    All packets sent
```

Nping with flags

Mandiant discovered that it is also possible to use a tool such as nping (or hping) to detect this variant of the Cisco implant.

REQUIREMENTS

- nping (installed with nmap)

SPEED

Class C - 256 IP addresses (4 hosts up) - 257.27 seconds

COMMAND LINE

nping -c1 -v3 --tcp -p 80 --seq 791104 --ack 3 10.1.1.1

FLAG EXPLANATION:

-c = count
-v = verbosity level
--tcp = TCP probe mode
-p = port
--seq = sequence number
--ack = acknowledge number
-H = (optional) Hide sent, which can speed up the scan

FIGURE 13: Using nping to detect the backdoor

OUTPUT

```
nping -c1 -v3 --tcp -p 80 --seq 791104 --ack 3 10.1.1.1

Starting Nping 0.6.47 ( http://nmap.org/nping ) at 2015-07-14 16:08 EDT
SENT (0.0048s) TCP [10.1.1.3:37895 > 10.1.1.1:80 S seq=791104 ack=3 off=5 res=0 win=1480
csum=0xED6E urp=0] IP [ver=4 ihl=5 tos=0x00 iplen=40 id=8373 foff=0 ttl=64 proto=6 csum=0x4416]
0000   45 00 00 28 20 b5 00 00  40 06 44 16 0a 01 01 03  E..(....@.D.....
0010   0a 01 01 01 94 07 00 50  00 0c 12 40 00 00 00 03  .......P...@....
0020   50 02 05 c8 ed 6e 00 00                           P....n..
RCVD (0.0092s) TCP [10.1.1.1:80 > 10.1.1.3:37895 SA seq=3 ack=791105 off=8 res=0 win=8192
csum=0x9256 urp=0 <mss 1460,nop,nop,sackOK,nop,wscale 5>] IP [ver=4 ihl=5 tos=0x00 iplen=52
id=18496 foff=0 ttl=255 proto=6 csum=0x5d7e]
0000   45 00 00 34 48 40 00 00  ff 06 5d 7e 0a 01 01 01  E..4H@....]~....
0010   0a 01 01 03 00 50 94 07  00 00 00 03 00 0c 12 41  .....P.........A
0020   80 12 20 00 92 56 00 00  02 04 05 b4 01 01 04 02  .....V..........
0030   01 03 03 05                                       ....
```

Highlighted areas include the sequence and acknowledge numbers. The difference must equal 791102, as well as the TCP flag options described above, which must be: "`20 04 05 b4 01 01 04 02 01 03 03 05`".

MITIGATION

After confirming compromise, the most effective mitigation is to reimage the router with a known clean download from Cisco. Ensure the new image hash values match and then harden the device to prevent future compromise. As mentioned in the overview, initial compromise is believed to occur due to either default or discovered credentials. After fixing the routers, focus on the rest of the network. If the router did not have default credentials, then the infection must have occurred some other way. A compromise assessment should follow.

APPENDIX A: COMPILED SNORT SIGNATURE (TCP SYN CONNECTIONS)

```
#include "sf_snort_plugin_api.h"
#include "sf_snort_packet.h"

/* declare detection functions */
int rule201504230eval(void *p);

/* declare rule data structures */
/* flow:to_server; */
static FlowFlags rule201504230flow0 =
{
   FLOW_TO_SERVER
};

static RuleOption rule201504230option0 =
{
   OPTION_TYPE_FLOWFLAGS,
   {
      &rule201504230flow0
   }
};

/* references for sid 201504230 */
static RuleReference *rule201504230refs[] =
{
   NULL
};

/* metadata for sid 201504230 */
/* metadata:; */
static RuleMetaData *rule201504230metadata[] =
{
   NULL
};

RuleOption *rule201504230options[] =
{
   &rule201504230option0,
   NULL
};

Rule rule201504230 = {
   /* rule header, akin to => tcp any any -> any any */
   {
      IPPROTO_TCP, /* proto */
      "any", /* SRCIP     */
      "any", /* SRCPORT   */
      0, /* DIRECTION */
      "any", /* DSTIP     */
      "any", /* DSTPORT   */
   },
   /* metadata */
   {
      3,  /* genid */
      201504230, /* sigid */
```

```
          1, /* revision */
       “misc-activity”, /* classification */
       0,  /* hardcoded priority */
       “TCP Trigger SEQ SYN”,     /* message */
       rule201504230refs, /* ptr to references */
       rule201504230metadata /* ptr to metadata */
   },
   rule201504230options, /* ptr to rule options */
   &rule201504230eval, /* use custom detection function */
   0 /* am I initialized yet? */
};

/* detection functions */
int rule201504230eval(void *p) {
   const u_int8_t *cursor_normal = 0;
   SFSnortPacket *sp = (SFSnortPacket *) p;
   uint32_t seq = 0;
   uint32_t ack = 0;

   if(sp == NULL)
      return RULE_NOMATCH;

   if(sp->payload == NULL)
      return RULE_NOMATCH;

   ack = ntohl(sp->tcp_header->acknowledgement);
   seq= ntohl(sp->tcp_header->sequence);

   if (ack == 0){
      return RULE_NOMATCH;
   }

   //Test for SYN packets
   if((sp->tcp_header->flags & TCPHEADER_SYN)&& !(sp->tcp_header->flags & TCPHEADER_ACK)){
      if ((ack > seq) && (ack - seq == 0xC123D)){ return RULE_MATCH; }
      else if ((seq > ack) && (seq - ack == 0xC123D)){ return RULE_MATCH;}
   }

   return RULE_NOMATCH;
}
```

APPENDIX B:
COMPILED SNORT SIGNATURE (TCP SYN-ACK CONNECTIONS)

```
#include “sf_snort_plugin_api.h”
#include “sf_snort_packet.h”

/* declare detection functions */
int rule201504231eval(void *p);

/* declare rule data structures */
/* flow:to_server; */
static FlowFlags rule201504231flow0 =
{
   FLOW_TO_SERVER
};

static RuleOption rule201504231option0 =
{
   OPTION_TYPE_FLOWFLAGS,
   {
      &rule201504231flow0
   }
};

/* references for sid 201504230 */
static RuleReference *rule201504231refs[] =
{
   NULL
};

/* metadata for sid 201504230 */
/* metadata:; */
static RuleMetaData *rule201504231metadata[] =
{
   NULL
};

RuleOption *rule201504231options[] =
{
   &rule201504231option0,
   NULL
};

Rule rule201504231 = {
   /* rule header, akin to => tcp any any -> any any */
   {
      IPPROTO_TCP, /* proto */
      “any”, /* SRCIP     */
      “any”, /* SRCPORT   */
      0, /* DIRECTION */
      “any”, /* DSTIP     */
      “any”, /* DSTPORT   */
   },
   /* metadata */
   {
```

```
3,  /* genid */
      201504231, /* sigid */
      1, /* revision */
      “trojan-activity”, /* classification */
      0,  /* hardcoded priority */
      “TCP Trigger SEQ SYN/ACK”,     /* message */
      rule201504231refs, /* ptr to references */
      rule201504231metadata /* ptr to metadata */
   },
   rule201504231options, /* ptr to rule options */
   &rule201504231eval, /* use custom detection function */
   0 /* am I initialized yet? */
};

/* detection functions */
int rule201504231eval(void *p) {
   const u_int8_t *cursor_normal = 0;
   SFSnortPacket *sp = (SFSnortPacket *) p;
   uint32_t seq = 0;
   uint32_t ack = 0;

   if(sp == NULL)
      return RULE_NOMATCH;

   if(sp->payload == NULL)
      return RULE_NOMATCH;

   ack = ntohl(sp->tcp_header->acknowledgement);
   seq= ntohl(sp->tcp_header->sequence);

   //Test for SYN/ACK packets
   if((sp->tcp_header->flags & TCPHEADER_SYN) && (sp->tcp_header->flags & TCPHEADER_ACK)){

      if ((ack > seq) &&  (ack - seq != 0xC123E)){ return RULE_NOMATCH; }
      else if ((seq > ack) && (seq - ack != 0xC123E)){ return RULE_NOMATCH; }

      //Hardcoded SYN/ACK has 6 options
      if (sp->num_tcp_options != 6) return RULE_NOMATCH;
      //MSS
      if(sp->tcp_options[0].option_code != 2) return RULE_NOMATCH;
      //NOP
      if(sp->tcp_options[1].option_code != 1) return RULE_NOMATCH;
      //NOP
      if(sp->tcp_options[2].option_code != 1) return RULE_NOMATCH;
      //SACK
      if(sp->tcp_options[3].option_code != 4) return RULE_NOMATCH;
      //NOP
      if(sp->tcp_options[4].option_code != 1) return RULE_NOMATCH;
      //Window Scale
      if(sp->tcp_options[5].option_code != 3) return RULE_NOMATCH;
      //compare the entire TCP options section
      if (memcmp(sp->tcp_options[0].option_data - 2,
          “\x02\x04\x05\xb4\x01\x01\x04\x02\x01\x03\x03\x05”, 12) != 0)
          return RULE_NOMATCH;

      //All conditions satisfied
      return RULE_MATCH;
   }
   return RULE_NOMATCH;
}

/*
Rule *rules[] = {
    &rule201504231,
    NULL
};
*/
```

About Mandiant

Mandiant, a FireEye company, has driven threat actors out of the computer networks and endpoints of hundreds of clients across every major industry. We are the go-to organization for the Fortune 500 and government agencies that want to defend against and respond to critical security incidents of all kinds. When intrusions are successful, Mandiant's security consulting services—backed up by threat intelligence and technology from FireEye—help organizations respond and resecure their networks.

About FireEye

FireEye protects the most valuable assets in the world from those who have them in their sights. Our combination of technology, intelligence, and expertise — reinforced with the most aggressive incident response team — helps eliminate the impact of security breaches. We find and stop attackers at every stage of an incursion. With FireEye, you'll detect attacks as they happen. You'll understand the risk these attacks pose to your most valued assets. And you'll have the resources to quickly respond and resolve security incidents. FireEye has over 3,100 customers across 67 countries, including over 200 of the Fortune 500.

For more information, please contact synfulknock@fireeye.com

For more about a Mandiant Compromise Assessment from FireEye, visit **www.fireeye.com/services.html**

MANDIANT®
A FireEye® Company

Mandiant, a FireEye Company | 703.683.3141 | 800.647.7020 | info@mandiant.com | **www.mandiant.com** | **www.fireeye.com**