

SYNful Knock - A Cisco router implant - Part I

September 15, 2015 | By [Bill Hau](#), [Tony Lee](#) | [Threat Research](#), [Advanced Malware](#)



Overview

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.

Webinar: SYNful Knock—A Cisco Router Implant

Fri, Sept. 18, 2015

[Register Now](#)

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 in order 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 article 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.

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 are enabled via the HTTP protocol (not HTTPS), using a specifically crafted TCP packets sent to the routers interface. The packets have a nonstandard sequence and corresponding acknowledgment numbers. The modules can manifest themselves as independent executable code or hooks within the routers IOS that provide functionality similar to the backdoor password. The backdoor password provides access to the router through the console and Telnet.



Figure 1: Module Updates

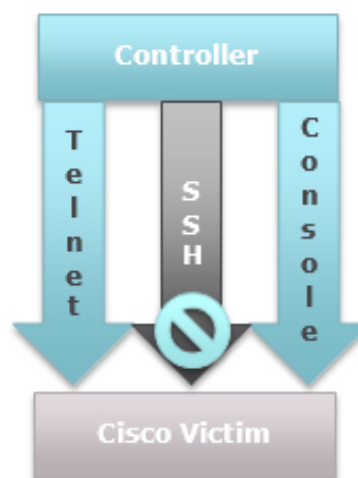


Figure 2: Backdoor Access

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].

Details

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

1. Modify the translation lookaside 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

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, then 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 set 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.

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	

Figure 3: Modification to TLB attributes

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.

16M	0xX0000000:0xXXXXFFFF	0xX0000000:0xXXXXFFFF	CacheMode=2, RW, Valid
16M	0xXX000000:0xXXXXFFFF	0xXX000000:0xXXXXFFFF	CacheMode=2, RW, Valid
16M	0xX0000000:0xXXXXFFFF	0x00000000:0xXXXXFFFF	CacheMode=3, RO, Valid
4M	0xXX000000:0xXXXXFFFF	0x0X000000:0xXXXXFFFF	CacheMode=3, RO, Valid
256K	0xXXX00000:0xXXXXFFFF	0x0XX00000:0xXXXXFFFF	CacheMode=3, RO, Valid
256K	0xXXX00000:0xXXXXFFFF	0xXXX00000:0xXXXXFFFF	CacheMode=3, RO, Valid
256K	0xXXX00000:0xXXXXFFFF	0XXX00000:0xXXXXFFFF	CacheMode=3, RW, Valid
256K	0xXXX00000:0xXXXXFFFF	0XXXX0000:0xXXXXFFFF	CacheMode=3, RW, Valid

Figure 4: TLB entries for a legitimate IOS image

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

16M	0xX0000000:0xXXXXFFFF	0xX0000000:0xXXXXFFFF	CacheMode=2, RW, Valid
16M	0xXX000000:0xXXXXFFFF	0xXX000000:0xXXXXFFFF	CacheMode=2, RW, Valid
16M	0xX0000000:0xXXXXFFFF	0x00000000:0xXXXXFFFF	CacheMode=3, RW, Valid
4M	0xXX000000:0xXXXXFFFF	0x0X000000:0xXXXXFFFF	CacheMode=3, RW, Valid
256K	0xXXX00000:0xXXXXFFFF	0x0XX00000:0xXXXXFFFF	CacheMode=3, RW, Valid
256K	0xXXX00000:0xXXXXFFFF	0XXXX0000:0xXXXXFFFF	CacheMode=3, RW, Valid
256K	0xXXX00000:0xXXXXFFFF	0XXX00000:0xXXXXFFFF	CacheMode=3, RW, Valid
256K	0xXXX00000:0xXXXXFFFF	0XXXX0000:0xXXXXFFFF	CacheMode=3, RW, Valid

Figure 5: TLB entries for a modified IOS image

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.

Initialize the Malware



To execute the malware during IOS image loading, Mandiant believes that 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 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, the behavior is such that it 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):...".

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.

Contact Us:
(877) 347-3393



Menu

plant also
of compromise that
can be used for detection purposes. The legitimate strings that are overwritten are shown in Figure 6.

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	65Next update
XXXXXXXX	20	64	75	65	20	69	6E	20	00	00	00	00	0A	43	75	72	due inCur
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..

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

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, 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 after you suspect you have been compromised.

XXXXXXXX	00 00 00 00 00 00 00 00 00 00 00 00 00 48 54 54 50HTTP
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-lubuntu7.7.
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***...

Figure 7: Malware strings

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 may vary depending on deployment and must first be discovered.

A modified IOS image will produce a very different and suspicious result when running what would seem to be an ordinary IOS command.

<html><body><div>.3.5-lubuntu7.7
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 to see if 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 ensures that the least amount of suspicion is raised. The following three instances were verified to enable access using the backdoor password:

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

Table 1: Authentication functions in which the secret backdoor password can be used

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 on compromise.

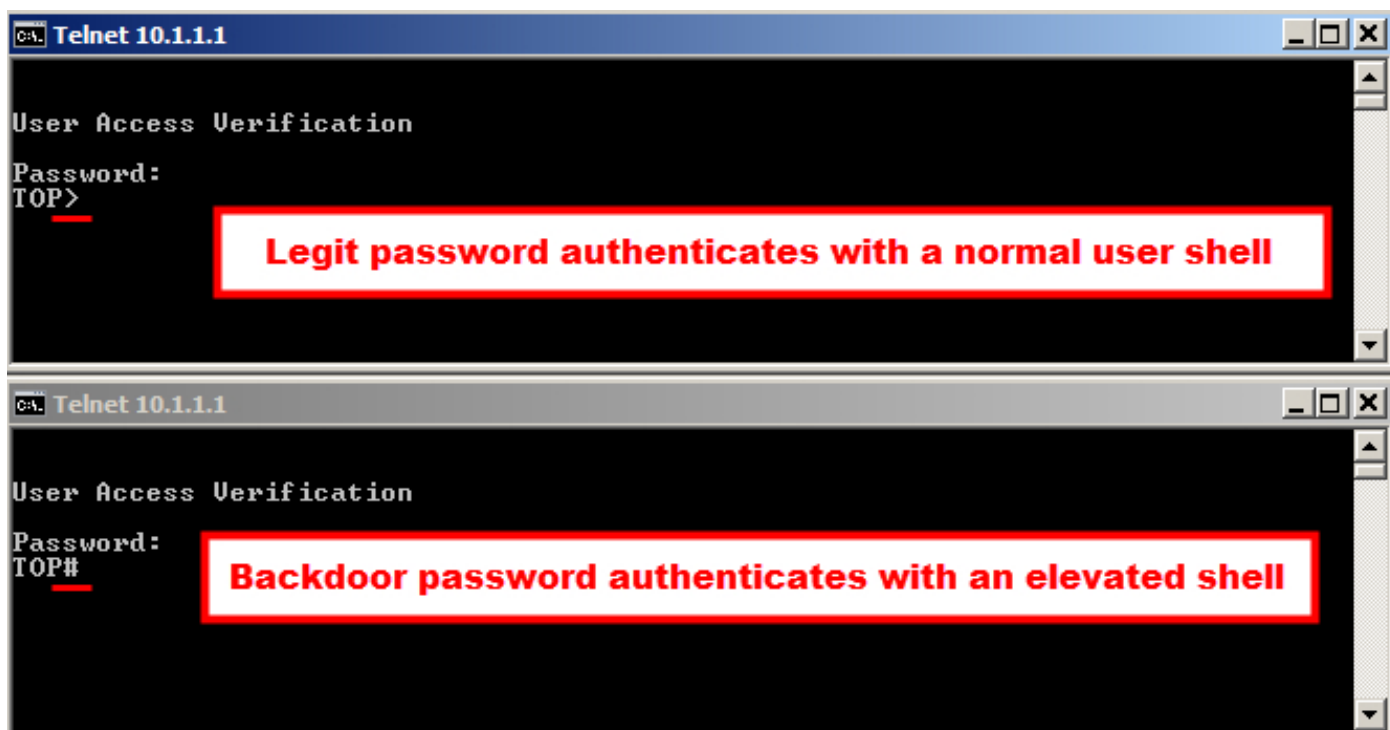


Figure 8: Subtle difference between authenticating using a legitimate password and the backdoor password

Network Command and Control (CnC)

The Command and Control 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 to 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.

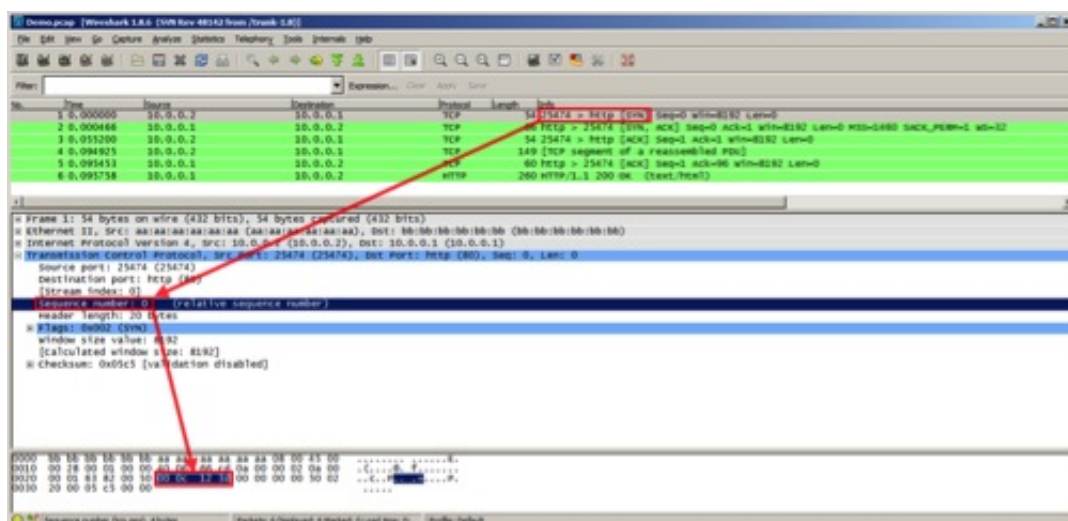


Figure 9: TCP SYN with sequence and acknowledgement offset of 0xC123D

1. 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.

These unique conditions are the anomalies that Mandiant used to write network detection signatures and tools.

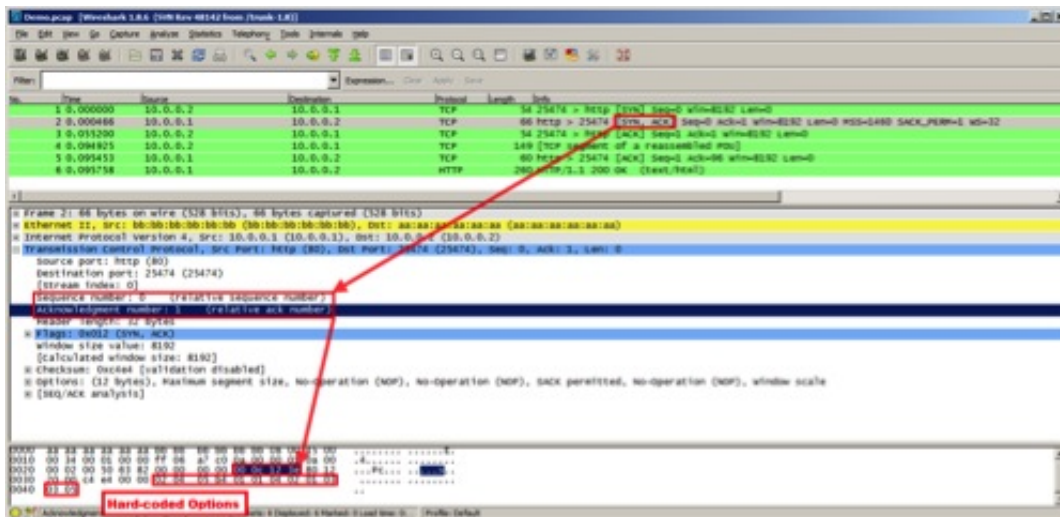


Figure 10: TCP SYN-ACK with sequence and acknowledgement offset of 0xC123E

1. 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].

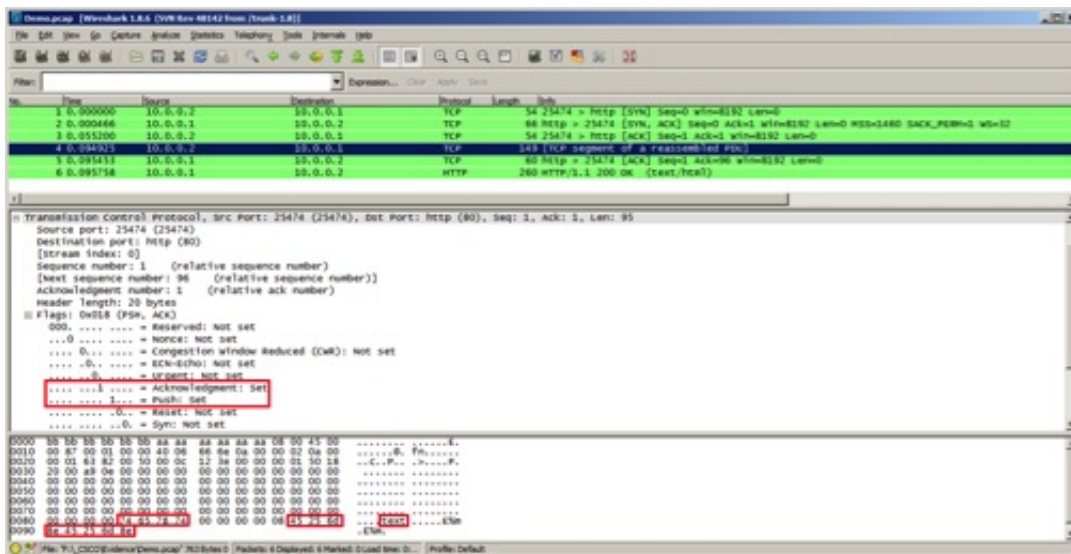


Figure 11: Controller command packet

1. The malware response is encapsulated in the following static HTTP/HTML server response. Fortunately, the response from the malware is not XOR encoded and will be easy to decipher.

```
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>
```

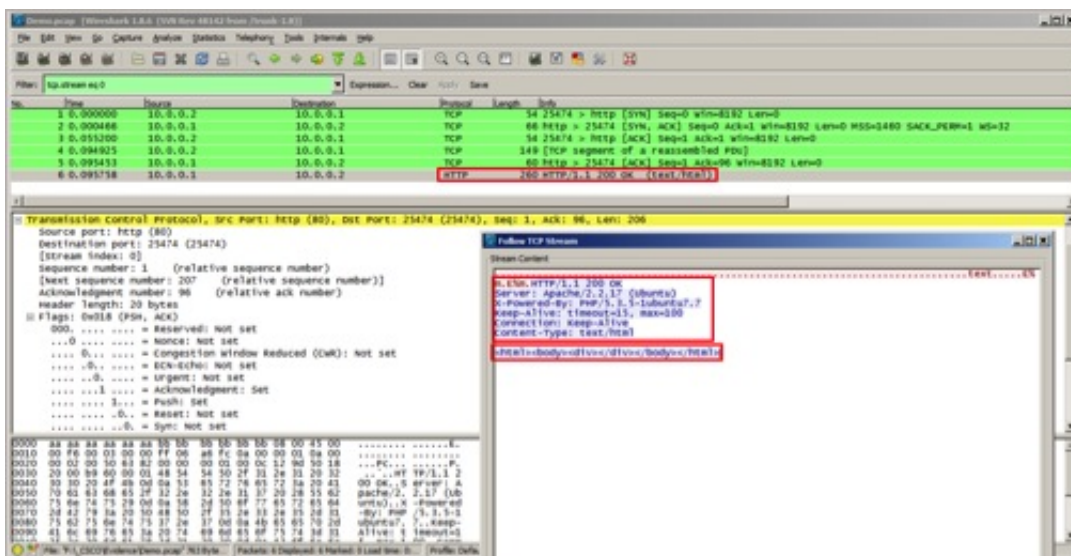


Figure 12: Victim response

Supported Commands

We mentioned previous that this implant is modular. The five commands shown in the table below are used for loading additional modules and functionality on the victim 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:

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.

Valid states are as follows:

- 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).

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.

	<p>Populate the memory allocated for the module. This command is used to populate the executable code and suspected configuration data.</p> <pre>[0x80 Bytes hook data][WORD first buffer length][WORD second buffer length] [First buffer...][Second buffer...]</pre> <p>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.</p> <p>After executing this command, the module state is set to one.</p>
3	<p>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.</p> <p>After executing this command, the module state is set to two.</p>
4	<p>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.</p>

Table 2: Supported Commands

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.

Conclusion

We hope that this post has advanced the understanding of this flexible and stealthy router implant. It should be evident now that this attack vector is very much a reality and will most likely grow in popularity and prevalence. In the next part of this series, we will examine methods that can be used to passively and actively detect this implant.

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

This entry was posted on Tue Sep 15 03:00:00 EDT 2015 and filed under [Advanced Malware](#), [Bill Hau](#), [Blog](#), [Latest Blog Posts](#), [Threat Research](#) and [Tony Lee](#).



FireEye Alerts

Be the first to receive information on major cyber attacks from the industry leader!



[Cyber Security Fundamentals](#)

[Careers](#)

[Events](#)

[Webinars](#)

[Support](#)

[Partners](#)

[Newsroom](#)

[Blog](#)

[Investor Relations](#)

[Incident?](#)

[Contact Us](#)

[Communication Preferences](#)

[Report Security Issue](#)

[Supplier Documents](#)

Connect



[Facebook](#)



[LinkedIn](#)



[Twitter](#)



[Google+](#)



[YouTube](#)



[Glassdoor](#)

Copyright © 2015 FireEye, Inc. All rights reserved.

[Privacy & Cookies Policy](#) | [Safe Harbor](#)