Overview

What is system() function?

The system() is an internal function to execute system related commands.

The commands could be removed or added without any notice.

Command Format

The format of the system() function is as follows:

```
string system(string $command_string[, string arg1, string arg2, ...]);
```

The system function returns a string after operation.

Format 1: only a command string without parameter

The followings are the examples which have a command string without any parameter.

```
<?php
system("php main.php");  // Run main.php
?>
```

```
<?php
system("php -d 3 main.php");  // Run main.php (restart delay: 3 seconds)
?>
```

```
<?php
// Run main.php (CPU time: 500us, restart delay: 3 seconds)
system("php -t 500 -d 3 main.php");
?>
```

Format 2: command with parameter(s)

The words starting with '%' followed by a number in the command string are replaced by parameters. This format is useful when a command includes any space or control character. The followings are the examples which have a command string with parameters.

```
<?php
$script = "main.php";
system("php %1",$script);  // Run main.php
?>
```

```
<?php
$delay = "3";
$script = "main.php";
```

```
system("php -d %1 %2", $delay, $script);  // Run main.php (restart delay: 3 seconds)
?>
```

```
<?php
$php_id = "0";
$cpu_time = "500";
$delay = "3";
$script = "main.php";
// Run main.php (CPU time: 500us, restart delay: 3 seconds)
system("php -t %2 -d %3 %4", $php_id, $cpu_time, $delay, $script);
?>
```

Control and Information Commands

uname command

This command returns PHPoC version, processor information, and hardware information. It requires a parameter string starting with '-' and supports multiple parameters with a parameter character.
Example: system("uname -sv");

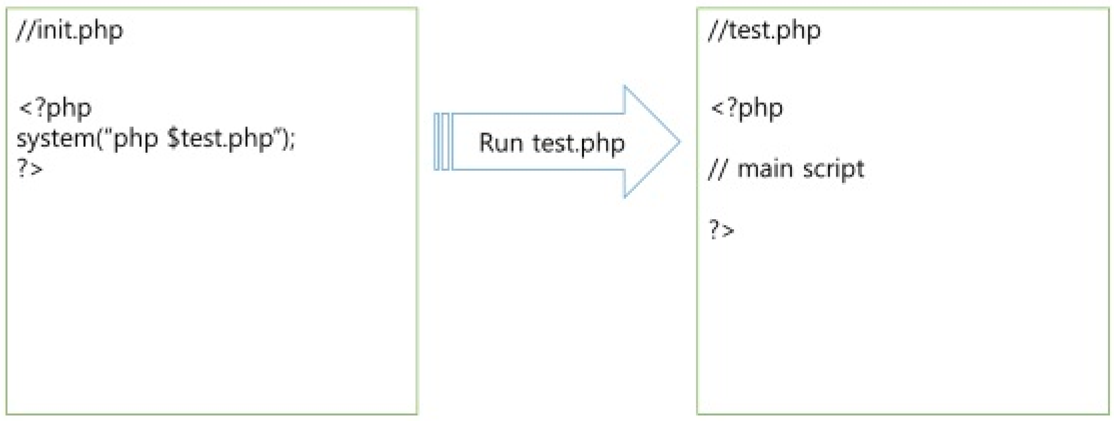
Parameter	Description
s	PHPoC engine name
v	PHPoC engine version
p	Processor information
i	Hardware platform information

The following is an example of the uname command.

```
<?php
// when the product is PBH-101
echo system("uname -s"), "₩r₩n";  // OUTPUT: PHPoC
echo system("uname -v"), "₩r₩n";  // OUTPUT: 1.0.0
echo system("uname -i"), "₩r₩n";  // OUTPUT: PBH-101
echo system("uname -p"), "₩r₩n";  // STM32F407 Cortex-M4F 168MHz
echo system("uname -svpi"), "₩r₩n";
// OUTPUT: PHPoC 1.0.0 STM32F407 Cortex-M4F 168MHz PBH-101
?>
```

php command

When the system boots up, the init.php is run in the beginning. User can program the one's code in the init.php and user can run other file with a "php" system command as well. In addition, user can adjust the running time per a loop of the task with this command.



```
//init.php

<?php
system("php $test.php");
?>
```

```
//test.php

<?php

// main script

?>
```

The following is the php command format.

- string system(“php [-t $cpu_time -d $restart_delay $script_name]”);

This command returns the script name to be run.

Parameter	Description
$cpu_time	$cpu_time is CPU running time per a loop of the task. The default value is 500us if it is omitted (range: 10 ~ 10000) If this value is larger, CPU occupation rate will be higher. But there might be network data losses when network load is high if this parameter is high.
$restart_delay	The script will be restarted automatically after this parameter time when the PHP script has been terminated. If this value is 0 the script will not be restarted. If it is omitted the default value is 5. The unit for this parameter is second. The script will not be restarted when the PHPoC Board is connected to the debugger
$script_name	If the $script_name is used for a parameter, this script will be restarted if the script is terminated. If it is omitted, the CPU time of the current task will be changed immediately.

The following example is an example which start a script when the current script is terminated.

- system("php test.php"); // run test.php after finishing this script

reboot command

User can restart PHPoC script or the system with the "reboot" command. The format of the reboot command is followed:

- string system(“reboot %1 [ms]”, $target);

The operations of this command are different depending on the $target as follows:

$target	Description
$php	restarts the PHPoC engine
$sys	restarts the product(system)

The [ms] is delay time and its unit is millisecond. If the [ms] is omitted the default value is 100 millisecond. This command returns the delay time.

```
<?php
echo "Restart PHPoC in 1 second...₩r₩n";
system("reboot php 1000");

while(1);
?>
```

Flash Command

The environmental variable area

There is some information which should be kept while system power is off. The flash memory is a non-volatile memory so it is a very good choice in this case.
There are a system data area for the system and a user data area for user data in the flash memory of PHPoC.

Data area	Description	Usage
system data area (/mmap/envs)	the area which values for system are saved into	network related data, other system data
user data area (/mmap/envu)	the area for user data	user data

There is only user data area information in this document. Please refer to other document for the system data area.

nvm command

nvm command

To save user data to the flash memory, save the data with a "nvm write" command after getting a key with a "nvm wkey" command.

- system("nvm wkey %1", $target);

After generating a key to be used for the "nvm write" command as a parameter, returns it.

Parameter	Description
$target	the area to save into (envs: system data area, envu: user data area)

- system("nvm write envs/envu wkey env");

Saving the data to the flash area with the key which was generated with "nvm wkey".

Parameter	Description
envs/envu	envs - system data area envu - user data area
wkey	the key which was returned from the "nvm wkey" command
env	data to be saved

After user have saved data to the flash memory user cannot save data in the same area within 2 seconds. And the number of saving operation is limited because of hardware limitation so this function should be carefully used.

The following is an example which saves "abcdefghij" to the user data area.

```
<?php
$str = "abcdefghij";

echo "setup /mmap/envu (user non-volatile meory)₩r₩n";
$wkey = system("nvm wkey envu");
echo "write ₩$str to /mmap/envu₩r₩n";
system("nvm write envu $wkey %1", $str);  // write $str  to /mmap/envu (flash)

echo "open /mmap/envu and read it₩r₩n";
$pid_envu = pid_open("/mmap/envu");  // open /mmap/envu
$buf = "";
pid_read($pid_envu, $buf, 10);  // read /mmap/envu
echo "/mmap/envu : $buf₩r₩n";

while(1);
?>
```

Crypto Commands

Encryption/Decryption

RC4

The RC4 is a stream cipher made by Ron Rivest, which is used in TLS and WEP. Because RC4 is an symmetric crypto algorithm both encryption and decryption commands are same.

The following is the methods of encryption and decryption in PHPoC.

- system("rc4 init %1", $rc4_key);

This command is for initializing PHPoC's RC4 crypto engine. So this command should be executed before the encryption/decryption. This command returns a context which is used for encryption and decryption operation.

Parameter	Description
$rc4_key	key for the RC4

- system("rc4 crypt %1 %2", $rc4, $rc4_text);

The RC4 encryption or decryption is performed with this command. It returns the data which was encrypted or decrypted.

Parameter	Description
$rc4	the context when the crypto engine was initialized
$rc4_text	plain text to be encrypted or cipher text to be decrypted

- system("rc4 skip %1", $rc4);

This command skips encryption or decryption operation. The skipping operation should be performed to improve the weakness of the RC4.

Parameter	Description
$rc4	the context when the crypto engine was initialized

The following is an example of the RC4 encryption and decryption.

```
// encryption
$rc4 = system("rc4 init %1", $rc4_key);  // initialize
$out = system("rc4 crypt %1 %2", $rc4, $rc4_pt);  // encryption

// decryption test
$rc4 = system("rc4 init %1", $rc4_key);  // initialize
$out = system("rc4 crypt %1 %2", $rc4, $rc4_ct);  // decryption
```

DES

The DES(Data Encryption Standard) is a symmetric key algorithm. And the triple DES gives stronger data security because it performs DES operation 3 times. There are ECB and CBC methods in the DES.

The following is the ECB data encryption/decryption algorithm.

- system("des init ecb/ede3_ecb enc/dec %1", $ecb_key);

This command is for initializing PHPoC's DES crypto engine. So this command should be executed prior to the encryption/decryption operation. This command returns a context which is used for encryption and decryption.

Parameter	Description
ecb/ede3_ecb	ecb - DES ECB ede3_ecb - 3DES ECB
enc/dec	enc - encryption dec - decryption
$ecb_key	64 bits-key to be used during the ECB encryption and decryption

- system("des crypt %1 %2", $des, $text);

The DES encryption or decryption is performed with this command. It returns the data which was encrypted or decrypted.

Parameter	Description
$des	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an example of the ECB-based DES encryption and decryption.

```
// encryption
$des = system("des init ecb enc %1", $ecb_key);  // initialize
$out = system("des crypt %1 %2", $des, $ecb_pt); // encryption

// decryption
$des = system("des init ecb dec %1", $ecb_key);  // initialize
$out = system("des crypt %1 %2", $des, $ecb_ct); // decryption
```

The following is an example of the ECB-based triple DES encryption and decryption.

```
// encryption
$des = system("des init ede3_ecb enc %1", $ecb_key);
$out = system("des crypt %1 %2", $des, $ecb_pt);

// decryption
$des = system("des init ede3_ecb dec %1", $ecb_key);
```

```
$out = system("des crypt %1 %2", $des, $ecb_ct);
```

The following is CBC data encryption/decryption algorithm.

- system("des init cbc/ede3_cbc enc/dec %1 %2", $cbc_key, $iv);

This command is for initializing PHPoC's DES crypto engine. So this command should be executed prior to the encryption/decryption operation. This command returns a context which is used for encryption and decryption.

Parameter	Description
cbc/ede3_cbc	cbc - DES CBC ede3_cbc - 3DES CBC
enc/dec	enc - encryption dec - decryption
$cbc_key	64 bits-key to be used during the ECB encryption and decryption
$iv	64 bits initialization vector

- system("des crypt %1 %2", $des, $text");

The DES encryption or decryption is performed with this command. It returns the data which was encrypted or decrypted.

Parameter	Description
$des	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an example of the CBC-based DES encryption and decryption.

```
// encryption
$des = system("des init cbc enc %1 %2", $cbc_key, $cbc_iv);  // initialize
$out = system("des crypt %1 %2", $des, $cbc_pt);              // encryption

// decryption
$des = system("des init cbc dec %1 %2", $cbc_key, $cbc_iv);  // initialize
$out = system("des crypt %1 %2", $des, $cbc_ct);              // decryption
```

aes command

The AES(Advanced Encryption Standard) is a widely-used crypto specification made by the NIST. There are ECB and CBC methods in the AES.

The following is an explanation to encrypt/decrypt data with the ECB.

- system("aes init ecb enc/dec %1", $ecb_key);

This command is for initializing PHPoC's AES crypto engine. So this command should be executed prior to the encryption/decryption. This command returns a context which is used for encryption and decryption.

Parameter	Description
enc/dec	enc - encryption dec - decryption
$ecb_key	the 128/192/256 bits key to be used encryption and decryption

- system("aes crypt %1 %2", $aes, $text);

The AES encryption or decryption is performed with this command according to initialization command. It returns the data which was encrypted or decrypted.

Parameter	Description
$aes	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an explanation to encrypt/decrypt data with the CBC.

- system("aes init cbc enc/dec %1 %2", $cbc_key, $iv);

This command is for initializing PHPoC's AES crypto engine. So this command should be executed prior to the encryption/decryption. This command returns a context which is used for encryption and decryption.

Parameter	Description
$enc/dec	enc - encryption / dec - decryption
$cbc_key	the 128/192/256 bits key to be used in encryption and decryption
$iv	128 bits initialization vector

- system("aes crypt %1 %2", $aes, $text);

The AES encryption or decryption is performed with this command according to initialization command. It returns the data which was encrypted or decrypted.

Parameter	Description
$aes	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an example of the CBC-based AES encryption and decryption.

```
// encryption
$aes = system("aes init cbc enc %1 %2", $cbc_key, $cbc_iv);
$out = system("aes crypt %1 %2", $aes, $cbc_pt16);

// decryption
$aes = system("aes init cbc dec %1 %2", $cbc_key, $cbc_iv);
$out = system("aes crypt %1 %2", $aes, $cbc_ct16);
```

seed command

The SEED is a widely-used crypto specification in South Korea made by the KISA. There are ECB and CBC methods in the SEED.

The following is an explanation to encrypt/decrypt data with the ECB.

- system("seed init ecb enc/dec %1", $type, $ecb_key);

This command is for initializing PHPoC's SEED crypto engine. So this command should be executed prior to the encryption/decryption operation. This command returns a context which is used for encryption and decryption.

Parameter	Description
enc/dec	enc - encryption dec - decryption
$ecb_key	the 128/256 bits key to used in encryption and decryption

- system("seed crypt %1 %2", $seed, $text);

The SEED encryption or decryption is performed with this command according to the initialization command. It returns the data which was encrypted or decrypted.

Parameter	Description
$seed	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an explanation to encrypt/decrypt data with the CBC.

- system("seed init cbc enc/dec %1 %2", $cbc_key, $iv);

This command is for initializing PHPoC's SEED crypto engine. So this command should be executed prior to the encryption/decryption. This command returns a context which is used for encryption and decryption.

Parameter	Description
$enc/dec	enc - encryption dec - decryption
$cbc_key	the 128/256 bits key to be used in encryption and decryption
$iv	128 bits initialization vector

- system("seed crypt %1 %2", $seed, $text);

The SEED encryption or decryption is performed with this command according to the initialization command. It returns the data which was encrypted or decrypted.

Parameter	Description
$seed	the context when the crypto engine was initialized
$text	plain text to be encrypted or cipher text to be decrypted

The following is an example of the CBC-based SEED encryption and decryption.

```
// encryption
$seed = system("seed init cbc enc %1 %2", $cbc_key, $cbc_iv);
$out = system("seed crypt %1 %2", $seed, $cbc_pt32);

// decryption
$seed = system("seed init cbc dec %1 %2", $cbc_key, $cbc_iv);
$out = system("seed crypt %1 %2", $seed, $cbc_ct32);
```

base64 command

The BASE64 is an encryption and decryption algorithm to convert binary data to ASCII, and vice versa. The BASE64 is used in email and XML. There are lots of alternations according to the usage. The PHPoC supports 3 types - standard type, URL type, and MIME type.

- system("base64 enc/dec %1 [std/mime/url]", $msg);

Parameter	Description
enc/dec	dec - encryption dec - decryption
$msg	the plain text to be encrypted or cipher text to be decrypted
std/mime/url	std - standard mime - MIME url - URL The default is standard if it is omitted.

The following is an example of BASE64.

```
$enc_out = system("base64 enc %1", $msg0);
$dec_out = system("base64 dec %1", $enc_out);
```

Hash

crc command

The crc command computes 8/16/32 bits CRC and its format is followed:

- system("crc bits %1 [init div msb/lsb]", $msg);

It returns CRC value after calculating with the parameters.

Parameter	Description
bits	8 - 8 bits CRC 16 - 16 bits CRC 32 - 32 bits CRC
$msg	the original message to be computed
init	CRC initial value. If it is omitted the default value is: 8 bits - ff, 16 bits - 1d0f, 32 bits - ffffffff
div	The divisor(polynomial) to be used for CRC calculation. If it is omitted the default value is: 8 bits - e0, 16 bits - 1021, 32 bits - edb88320
msb/lsb	the CRC calculation order msb: calculated from the MSB to LSB lsb: calculated from the LSB to MSB If it is omitted the default value is: 8 bits - lsb, 16 bits - msb, 32bits - lsb

The following is an example code for each CRC types.

```
<?php

$string = "123456789";

printf("CRC-16-ANSI : %04x₩r₩n", (int)system("crc 16 %1 0000 a001 lsb", $string));

printf("CRC-16-Modbus : %04x₩r₩n", (int)system("crc 16 %1 ffff a001 lsb", $string));

printf("CRC-CCITT FFFF: %04x₩r₩n", (int)system("crc 16 %1 ffff 1021 msb", $string));

printf("CRC-CCITT 1D0F: %04x₩r₩n", (int)system("crc 16 %1 1d0f 1021 msb", $string));

printf("CRC-CCITT XModem : %04x₩r₩n", (int)system("crc 16 %1 0000 1021 msb", $string));

$crc16_out = (int)system("crc 16 123456789 %1 8408 lsb", $string);
$crc16_out = bin2int(int2bin($crc16_out, 2, true), 0, 2);
printf("CRC-CCITT Kermit : %04x₩r₩n", $crc16_out);

$crc16_out = (int)system("crc 16 123456789 ffff 8408 lsb");
$crc16_out = $crc16_out ^ 0xffff;
printf("CRC-CCITT PPP     : %04x₩r₩n", $crc16_out);

$crc16_out = ~(int)system("crc 16 %1 0000 a6bc lsb", $string);
$crc16_out = bin2int(int2bin($crc16_out, 2, true), 0, 2);
```

```
printf("CRC-16-DNP         : %04x₩r₩n", $crc16_out);

?>
```