

# [REVS] Bypassing MSB Data Filters for Buffer Overflows on Intel Platforms

*Source:* <http://www.derkeiler.com/Mailing-Lists/Securiteam/2005-05/0164.html>

---

*From:* SecuriTeam ([support\\_at\\_securiteam.com](mailto:support_at_securiteam.com))

*Date:* 05/31/05

To: [list@securiteam.com](mailto:list@securiteam.com)

Date: 31 May 2005 17:55:31 +0200

The following security advisory is sent to the securiteam mailing list, and can be found at the SecuriTeam web site: <http://www.securiteam.com>

-- promotion

The SecuriTeam alerts list – Free, Accurate, Independent.

Get your security news from a reliable source.

<http://www.securiteam.com/maillinglist.html>

-----

Bypassing MSB Data Filters for Buffer Overflows on Intel Platforms

---

## SUMMARY

The short whitepaper below describes a technique to encode binary assembly code to printable characters. The technique described here is applicable for the x86 platform.

## DETAILS

### Introduction:

Buffer overflows aim to execute carefully chosen machine-native instructions on a target system. That code is a series of bytes that cross the full range of possible values. Unfortunately for many attackers, certain servers filter out or modify any values outside the range 21 to 7F hex. Examples are web proxies and e-mail servers that cannot handle non-printable ASCII values in their data. Their input filters mangle the incoming exploit code, and as a result destroy its functionality.

The author posed a challenge to several hackers one Saturday night, and this paper is the result. The algorithm presented here will encode any sequence of binary data into ASCII characters that, when interpreted by an Intel processor, will decode the original sequence and execute it.

## Securiteam: [REVS] Bypassing MSB Data Filters for Buffer Overflows on Intel Platforms

The process is fairly simple: move the stack pointer just past the ASCII code, decode 32-bits of the original sequence at a time, and push that value onto the stack. As the decoding progresses, the original binary series is "grown" toward the end of the ASCII code. When the ASCII code executes its last PUSH instruction, the first bytes of the exploit code are put into place at the next memory address for the processor to execute.

### Terminology:

Printable – Any byte value between 0x21 and 0x7f. For multi-byte objects like words, each composite byte must be printable for the object to be considered printable.

### What you need

A buffer filled with the exploit code to execute. Good examples for shellcodes can be found at <http://metasploit.org/> Metasploit.org

### Part I: Align exploit code

The exploit code *must* be aligned on a 32-bit boundary. Pre-pad or post-pad with NOPs to make it all tidy.

### Part II: Construct ASCII code

The Intel assembly instructions AND, SUB, PUSH, and POP are sometimes encoded as single-byte, printable instructions.

Specifically, we use only printable operands (e.g. 0x21212121 → 0x7f7f7f7f) and rely on these operations: (AND EAX, #####), (SUB EAX, #####), (PUSH EAX), (POP ESP). Using those operations, it is possible to set EAX to any value that we wish, set ESP to any value we wish, and thus set any value into any stack-addressable memory location.

### Step 1:

Clear EAX, as it is our only real "register" and it is critical to know its starting value.

```
AND EAX, 5e5e5e5e
```

```
AND EAX, 21212121
```

```
ASCII: %^^^^%!!!!
```

### Step 2, Option 1:

Set ESP to the other side of the bridge. In code, we'll need to put a placeholder here. The correct value of ESP will be `overflow_starting_address + ASCII_code_size + exploit_code_size`, which will not be known until we're done generating the ASCII code.

Once you have this address, put it into ESP like this:

```
SUB EAX, #####
```

```
SUB EAX, #####
```

```
PUSH EAX
```

```
POP ESP
```

```
ASCII: -****_****P\ (**** is a placeholder for later values)
```

## Securiteam: [REVS] Bypassing MSB Data Filters for Buffer Overflows on Intel Platforms

### Step 2, Option 2:

Alternatively, if you don't know the memory address where the overflow will occur, you can calculate the offset from ESP to the beginning of the exploit code and simply code SUB instructions to wrap ESP to the correct end-of-code address. Once you have the offset from the original ESP (see Step 4 below), adjust ESP like this:

```
PUSH ESP
POP EAX
SUB EAX, #####
SUB EAX, #####
PUSH EAX
POP ESP
ASCII: TX-****-****P\ (**** is a placeholder for later values)
```

### Step 3:

Create the units that will decode into exploit code... BACKWARD. Parse the last 32-bits first, and proceed toward the beginning of the exploit buffer. PUSH operates in the opposite direction that code executes, so here's where we reverse the process to correct for that.

```
SUB EAX, ##### (Using SUB, wrap EAX around until it
SUB EAX, ##### arrives at the value of the current 32-bit
SUB EAX, ##### section of your exploit code)
PUSH EAX
```

```
ASCII: -****-****-****P
```

..repeat as necessary...

### Step 4:

Now that the ASCII code array is generated, count its size in bytes, add the size of the exploit array, and add the memory address where the overflow will occur. Using the same technique as for exploit code, derive the values for Step 2 to replace the \*\*\*\* values.

### Part III: Inject ASCII code.

The Evil Empire's IDS won't know what hit it.

### Comments:

Yes, this makes a huge buffer to inject. Obviously this code is to be used sparingly when you really, really need it. On the other hand, very few IDS's will take note of an innocuous string of ASCII symbols in a username or password field. In fact, the packet may get a nice little pat on the back from the security system if this happens to be a password-field overflow. "Good job, user, for selecting a great password!"

### ADDITIONAL INFORMATION

The original article can be found at:

<<http://community.core-sdi.com/~juliano/bypass-msb.txt>>  
<http://community.core-sdi.com/~juliano/bypass-msb.txt>

=====

Securiteam: [REVS] Bypassing MSB Data Filters for Buffer Overflows on Intel Platforms

This bulletin is sent to members of the SecuriTeam mailing list.

To unsubscribe from the list, send mail with an empty subject line and body to:

list-unsubscribe@securiteam.com

In order to subscribe to the mailing list, simply forward this email to: list-subscribe@securiteam.com

=====  
=====

**DISCLAIMER:**

The information in this bulletin is provided "AS IS" without warranty of any kind.

In no event shall we be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.