

# VIEWSTATE VULNERABILITIES

TIMUR YUNUSOV, POSITIVE TECHNOLOGIES



1.	VIEWSTATE OVERVIEW .....	3
3.	PROTECTION .....	8
4.	CONCLUSION .....	9
5.	REFERENCE .....	10
6.	ABUOT POSITIVE TECHNOLOGIES .....	11

## 1. VIEWSTATE OVERVIEW

---

*"View state is a method that the ASP.NET page framework uses to preserve page and control values between round trips. When the HTML markup for the page is rendered, the current state of the page and values that must be retained during postback are serialized into base64-encoded strings. This information is then put into the view state hidden field or fields."*

MSDN

*"What does ViewState do?"*

- Stores values per control by key name, like a Hashtable
- Tracks changes to a ViewState value's initial state
- Serializes and deserializes saved data into a hidden form field on the client
- Automatically restores ViewState data on postbacks"

From an article on the ViewState mechanisms by an ASP.NET developer

To put it even simpler, ViewState is a hidden HTML parameter that sends a current structure of page content to the server. Example of use: retaining form field values on the page for by-page list scrolling.

Though there are widely used methods of disabling or avoiding ViewState (usually, by means of a DBMS), this mechanism is built in ASP.NET by default and is often misused:

*"Even more important than understanding what it does, is understanding what it does NOT do:*

*What doesn't ViewState do?"*

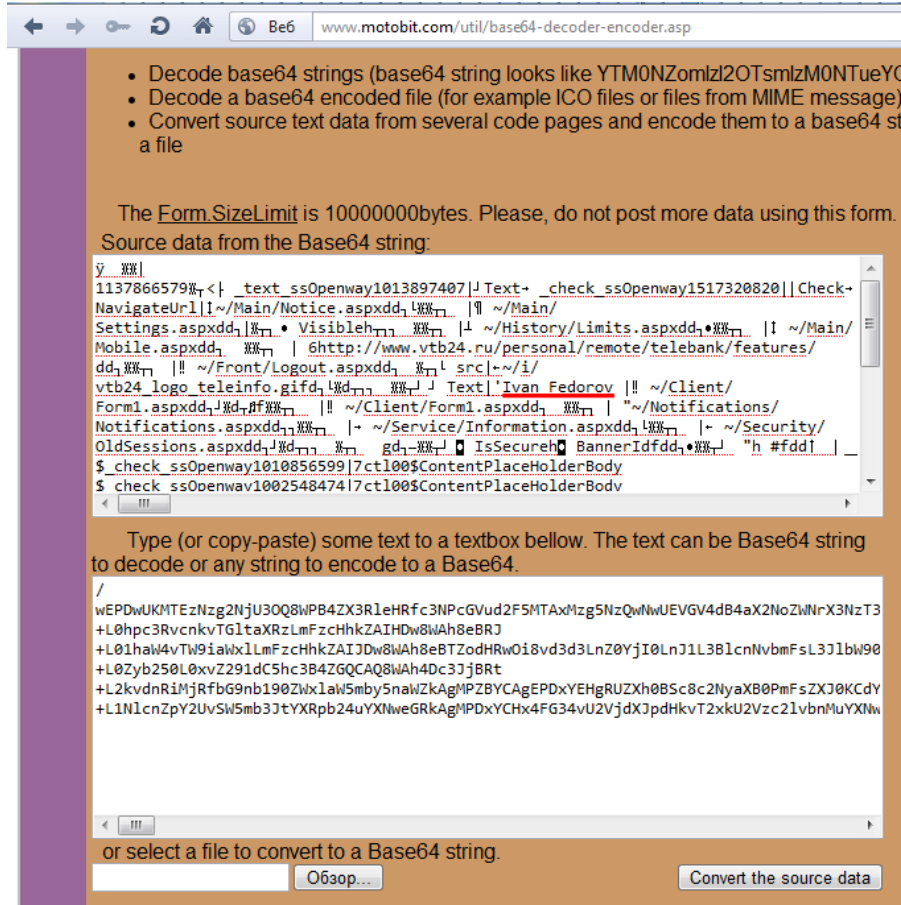
- Automatically retain state of class variables (private, protected, or public)
- Remember any state information across page loads (only postbacks)
- Remove the need to repopulate data on every request
- ViewState is not responsible for the population of values that are posted such as by TextBox controls (although it does play an important role)"

From an article on the ViewState mechanisms by an ASP.NET developer

Obviously, such misuse entails more serious problems, such as a missing filtration or a perverted idea of how the web application should work properly.

Developers tend to believe that if ViewState is a serialized structure, moreover, a base64-encoded one, no attacker will be able to get to its contents.

However, the truth is, if the encryption and the data integrity check (MAC) are disabled, accessing the content is much simpler than it seems. Let's decode base64:



- Decode base64 strings (base64 string looks like YTM0NZomlZl2OTsmlzM0NTueYC
- Decode a base64 encoded file (for example ICO files or files from MIME message)
- Convert source text data from several code pages and encode them to a base64 string file

The `Form.SizeLimit` is 10000000bytes. Please, do not post more data using this form.

Source data from the Base64 string:

```

y...
1137866579%<|_text_ssOpenway1013897407|JText+_check_ssOpenway1517320820||Check+
NavigateUrl|I~/Main/Notice.aspxdd,|...|~/Main/
Settings.aspxdd,|...• Visibleh...|~/History/Limits.aspxdd,•...|~/Main/
Mobile.aspxdd,|...|6http://www.vtb24.ru/personal/remote/telebank/features/
dd,|~/Front/Logout.aspxdd,|src|~/i/
vtb24_logo_teleinfo.gifd,|...|Text|Ivan Fedorov|~/Client/
Form1.aspxdd,|~#f...|~/Client/Form1.aspxdd,|~/Notifications/
Notifications.aspxdd,|~/Service/Information.aspxdd,|~/Security/
OldSessions.aspxdd,|gd,|IsSecureh BannerIdfdd,|h #fdd|
$_check_ssOpenway1010856599|7ct100$ContentPlaceHolderBody
$ check_ssOpenwav1002548474|7ct100$ContentPlaceHolderBody

```

Type (or copy-paste) some text to a textbox below. The text can be Base64 string to decode or any string to encode to a Base64.

```

/
wEPDwUKMTEzNzg2NjU3OQ8WPB4ZX3R1eHRfc3NpcGVud2F5MTAxMzg5NzQwNwUEVGV4dD4B4a2NoZWlNwX3NzT3
+L0hpc3RvcnkVtG1taXRzLmFzcHhkZAIHDw8NAh8eBRJ
+L01haW4vTW9iaWx1LmFzcHhkZAIJDw8NAh8eBTZodHRwOi8vd3d3LnZ0YjI0LnJlL3BlcnVbmFsL3JlLW90
+L0Zyb250L0xvZ291dC5hc3B4ZGQAQ8WAh4Dc3JjBRt
+L2kydnRiMjRfbG9nb190ZWxlalw5bW5naWZkaGMPZBYCAgEPDxYEHgRUZxh0BSc8c2NyaXB0PmFsZXJ0KCDy
+L1N1cnZpY2UvSw5mb3JtYXRpb24uYXNweGRkAgMPDxYCHx4FG34vU2VjdXJpdHkvT2xkU2VzL2lbnMuYXNw

```

or select a file to convert to a Base64 string.

content is simpler than it seems. decode

Then, open it in the Hex Editor. Now it is evident that any string variable is preceded by bytes that indicate the string's length (the number of bites depends on the length of the string: a string less than 128 bytes will have one byte for a variable length).

Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
00001280	64	64	02	0B	0F	0F	16	02	1F	1E	05	13	7E	2F	46	72	dd.....~/Fr
00001296	6F	6E	74	2F	4C	6F	67	6F	75	74	2E	61	73	70	78	64	ont/Logout.aspxd
00001312	64	02	01	0F	16	02	1E	03	73	72	63	05	1B	7E	2F	69	d.....src..~/i
00001328	2F	76	74	62	32	34	5F	6C	6F	67	Байт размера		65				/vtb24_logo_tele
00001344	69	6E	66	6F	2E	67	69	66	64	02	переменной.		02				info.gifd...d...
00001360	01	0F	0F	16	04	1E	04	54	65	78	74	05	27	3C	73	63	.....Text.'<sc
00001376	72	69	70	74	3E	61	6C	65	72	74	28	27	58	53	53	21	ript>alert('XSS!
00001392	27	29	3C	2F	73	63	72	69	70	74	3E	54	45	53	54	20	')</script>TEST
00001408	54	45	53	54	1F	1E	05	13	7E	2F	43	6C	69	65	6E	74	TEST....~/Client
00001424	2F	46	6F	72	6D	31	2E	61	73	70	78	64	64	02	04	0F	/Form1.aspxdd...
00001440	64	16	0E	66	0F	0F	16	02	1F	1E	05	13	7E	2F	43	6C	d..f.....~/Cl
00001456	69	65	6E	74	2F	46	6F	72	6D	31	2E	61	73	70	78	64	ient/Form1.aspxd
00001472	64	02	01	0F	0F	16	02	1F	1E	05	22	7E	2F	4E	6F	74	d....."/Not
00001488	69	66	69	63	61	74	69	6F	6E	73	2F	4E	6F	74	69	66	ifications/Notif
00001504	69	63	61	74	69	6F	6E	73	2E	61	73	70	78	64	64	02	ications.aspxdd.
00001520	02	0F	0F	16	02	1F	1E	05	1A	7E	2F	53	65	72	76	69	.....~/Servi
00001536	63	65	2F	49	6E	66	6F	72	6D	61	74	69	6F	6E	2E	61	ce/Information.a
00001552	73	70	78	64	64	02	03	0F	0F	16	02	1F	1E	05	1B	7E	spdd.....~/
00001568	2F	53	65	63	75	72	69	74	79	2F	4F	6C	64	53	65	73	/Security/OldSes
00001584	73	69	6F	6E	73	2E	61	73	70	78	64	64	02	04	0F	64	sions.aspxdd...d
00001600	16	02	02	01	0F	16	02	1F	1F	67	64	02	06	0F	0F	16	.....gd.....
00001616	04	1E	08	49	73	53	65	63	75	72	65	68	1E	08	42	61	..IsSecureh..Ba
00001632	6E	6E	65	72	49	64	66	64	64	02	07	0F	0F	16	04	1F	nnerIdfdd.....
00001648	22	68	1F	23	66	64	64	18	01	05	1E	5F	5F	43	6F	6E	"h.#fdd....__Con
00001664	74	72	6F	6C	73	52	65	71	75	69	72	65	50	6F	73	74	trolsRequirePost
00001680	42	61	63	6B	4B	65	79	5F	5F	16	0F	05	37	63	74	6C	BackKey.....7ctl
00001696	30	30	24	43	6F	6E	74	65	6E	74	50	6C	61	63	65	48	00\$ContentPlaceH
00001712	6F	6C	64	65	72	42	6F	64	79	24	5F	63	68	65	63	6B	olderBody\$_check

Authoritative resources state that ASP.NET versions earlier than 2.0 use LosFormatter as a serialization/deserialization algorithm, while version 2.0 and later use ObjectStateFormatter. Thus, to change the variable, one needs to define the length of a new string, overwrite the string, overwrite the byte (bytes) with the string length, encode it back with base64 and insert into \_\_VIEWSTATE.

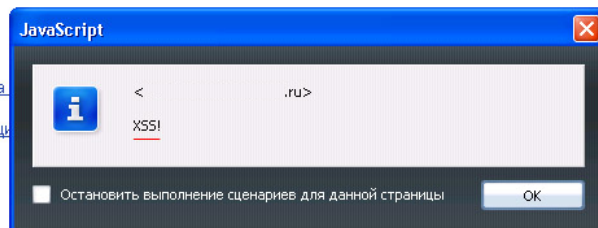


**EXPLOIT!**

[Счета и карты](#)

[Мои договоры](#)

[Оповещения](#) [Подписка](#)  
[вес](#) [Контактная информация](#)



## 2. VULNERABILITIES AND ATTACKS

---

Combined with a low-level knowledge of an average specialist about a correct and secure configuration of web applications, such approach generates the following vulnerabilities and provides opportunities for the following attacks:

- Cross-Site Scripting (XSS)
- Content Spoofing
- SQL Injection
- Information Leakage
- Logical Attacks
- ViewState Vulnerabilities as such
- Other vulnerabilities

### 2.1. Cross-Site Scripting, Content Spoofing

The possibility of content spoofing for an HTML page comes out of ViewState main purpose, i.e. to preserve page and control values. If data from ViewState placed into the HTTP response body are not filtered properly, it results in Content Spoofing and/or Cross-Site Scripting.

**Vulnerable configuration:**

**EnableViewStateMac=false**

**ViewStateEncryptionMode=never|auto**

*(Depends on RegisterRequiresViewStateEncryption)*

**ViewStateUserKey=EMPTY**

### 2.2. Information Leakage, Logical Attacks

If developer does not encrypt the VIEWSTATE parameter (Securing View State), an attacker can decode the VIEWSTATE structure and extract confidential data. If developer does not check data integrity (MAC), an attacker can change parameters that can influence the web application logic, thus facilitating Authentication Bypass, Authorization Bypass, and Abuse of Functionality.

**Vulnerable configuration:**

**ViewStateEncryptionMode=never|auto**

**EnableViewStateMac=false|true**

### 2.3. Attacks Against ViewState

The ViewState itself is also vulnerable to attacks. For example, September, 2010 saw a publication describing a vulnerability that allowed decrypting AES-encrypted ViewState by sending numerous requests to a server and tracking various error codes [1].

Besides, the earlier versions (1.0, 1.1) are vulnerable to the Denial Of Service (DoS) attacks (against unencrypted VIEWSTATE) and the Replay attacks (against encrypted VIEWSTATE). The latter one is an attack against a cryptographic protocol consisting in resending an intercepted package that will be received appropriately, thus breaking the algorithm. These attacks were described by Michal Zalewski as far as in 2005 [2].

## **2.4. Other Vulnerabilities**

All other vulnerabilities common for web applications, such as SQL injection, OS Commanding, as well as other vulnerabilities of such types as Code Exploitation, Information Disclosure, etc. can and should be checked both in variables of the ViewState structure and in ordinary variables sent by GET/POST/COOKIES.

### **Vulnerable configuration:**

**EnableViewStateMac=false**

**ViewStateEncryptionMode=never|auto**

***(depends on RegisterRequiresViewStateEncryption)***

## 3. PROTECTION

---

### 3.1. EnableViewStateMac

**Default: TRUE**

**Since: 1.0**

Enables MAC (Machine Authentication Check) to check the VIEWSTATE parameter values by means of a checksum.

Set the *EnableViewStateMac* property to "True" in the *Page* element.

Besides, the activation requires configuring the *validationKey* and *validation* properties of the *machineKey* element.

The following in-built encrypting algorithms are supported: SHA1, MD5, 3DES, AES, HMACSHA256, HMACSHA384, HMACSHA512.

### 3.2. ViewStateEncryptionMode

**Default: Auto**

**Since: 2.0**

Allows encrypting the VIEWSTATE parameter by any of the following algorithms: DES, 3DES, AES.

For activation, configure the *decryptionKey* and *decryption* properties of the *machineKey* element.

### 3.3. ViewStateUserKey

**Default: EMPTY**

**Since: 1.1**

Not everyone knows that ViewState protects not only itself against spoofing, but the entire application against CSRF by means of the *ViewStateUserKey* parameter.

*ViewStateUserKey* is just a protection mechanism. It is a developer's duty to ensure its unpredictable and random nature.

Set the *ViewStateUserKey* property to "String" in the *Page* element.

## 4. CONCLUSION

---

Sections 2 and 3 provide sound evidence that, configured by default, ViewState is secured against vulnerabilities that are not 0-day. However, quite often developers, after having struggled with constantly appearing error notifications about integrity violation, faulty arguments, etc., end up disabling keys that provoke errors, thus leaving the application vulnerable to various attacks.

Yet, if the web application is properly configured, the probability of errors and even vulnerabilities can be minimized down to 0.

## 5. REFERENCE

---

[1] <http://weblogs.asp.net/scottgu/archive/2010/09/18/important-asp-net-security-vulnerability.aspx>

[2] <http://seclists.org/bugtraq/2005/May/27>

## 6. ABOUT POSITIVE TECHNOLOGIES

---

**Positive Technologies** [www.ptsecurity.com](http://www.ptsecurity.com) is among the key players in the IT security market in Russia.

The principal activities of the company include the development of integrated tools for information security monitoring (**MaxPatrol**); providing IT security consulting services and technical support; the development of the Securitylab [en.securitylab.ru](http://en.securitylab.ru) leading Russian information security portal.

Among the clients of **Positive Technologies** are more than 40 state enterprises, more than 50 banks and financial organizations, 20 telecommunication companies, more than 40 plant facilities, as well as IT, service and retail companies from Russia, CIS countries, Baltic States, China, Ecuador, Germany, Great Britain, Holland, Iran, Israel, Japan, Mexico, South African Republic, Thailand, Turkey and USA.

**Positive Technologies** is a team of highly skilled developers, advisers and experts with years of vast hands-on experience. The company specialists possess professional titles and certificates; they are the members of various international societies and are actively involved in the IT security field development.