

## Security bugs discovered in Ubiquiti Unifi Controller

**Affected Component:** Testing was performed on v3.1.4 (Linux). Stable versions and other platforms may be affected as well.

**Credits:** Luca Caretoni

### #1 Insecure Java Random() to generate secret tokens - HIGH RISK

java.util.Random is used across the entire codebase to generate secret tokens, such as session cookies, AP auth keys and reset tokens. This class is not suitable for strong random strings generation. Under some circumstances, it seems practical to predict the reset password token and compromise the admin account, which would lead to full compromise of the entire platform.

In the Java version shipped with the Unifi controller, Random() depends on the time in nanosecond and a static seed.

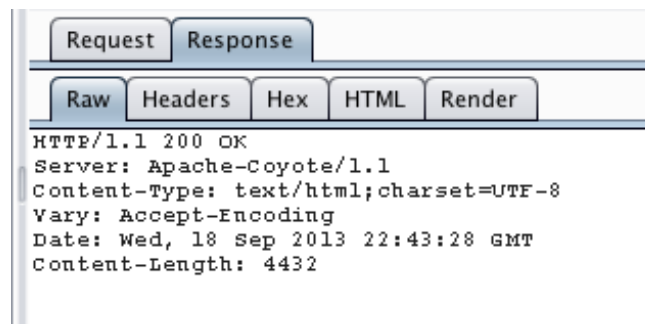
```
77 public Random() { this(++seedUniquifier + System.nanoTime()); }  
78 private static volatile long seedUniquifier = 8682522807148012L;
```

Considering that a single instance of Random() is used across the Unifi Controller and multiple method invokes nextInt() (thus modifying the seed state), the exploitability of this issue is not trivial.

In order to predict a token and compromise the application, an attacker would require:

- Admin email
- Time nanosecond of *com.ubnt.ace.E* class loading
- Exact sequence of previous usage (e.g. how many reset tokens have been generated)

A realistic attack can occur just after a reboot of the controller, as the application would be in a "clean" state. By invoking the recover password request and inspecting the HTTP response Date header, an attacker could predict with good approximation the time of the server



The screenshot shows a web browser's developer tools interface. At the top, there are two tabs: 'Request' and 'Response', with 'Response' selected. Below the tabs, there are five sub-tabs: 'Raw', 'Headers', 'Hex', 'HTML', and 'Render', with 'Raw' selected. The main content area displays the raw HTTP response text:

```
HTTP/1.1 200 OK  
Server: Apache-Coyote/1.1  
Content-Type: text/html; charset=UTF-8  
Vary: Accept-Encoding  
Date: Wed, 18 Sep 2013 22:43:28 GMT  
Content-Length: 4432
```

Having that information, it is feasible to bruteforce the actual Random seed used by the remote server, using the following pseudocode:

```

//Submit a "forgot your password" request and dump the response

long seedUniquifier = 8682522807148012L+1; //Static seed, plus one
Long nstime; //Time retrieved from the HTTP response Date header

for (int i=-99999; i < 999999; i++) {

    Random myRnd = new Random(seedUniquifier + nstime + i);

    //Using myRnd, generate a 16bit token from "abcdefghijklmnopqrstuvxyz" + "abcdefghijklmnopqrstuvxyz".toUpperCase()
    //Submit the token to GET /verify?email=<EMAIL>&t=<GENERATED TOKEN>

```

As the controller code does not limit the number of verify token requests, it is possible to validate the generated tokens. According to my preliminary testing, the attack is possible although it would require several hours to succeed.

Random() must be replaced with SecureRandom().

## #2 System-wise Cross Site Request Forgery - HIGH RISK

Unifi Controller does not protect the application against Cross-Site-Request-Forgery (CSRF) attacks. Please refer to [http://en.wikipedia.org/wiki/Cross-site\\_request\\_forgery](http://en.wikipedia.org/wiki/Cross-site_request_forgery).

For instance, it would be possible to override arbitrary options within `system.properties` that could eventually lead to full-compromise (e.g. by appending malicious MongoDB parameters).

The following is an example of a malicious HTML page that would create the key "CSRF = CSRF".

```

<html>
<body>
<form action="https://<CONTROLLER_IP>:8443/api/s/default/cmd/system" method="POST">
<input type="hidden" name="json" value="{&#123;&quot;cmd&quot;:&#58;&quot;set&#45;param&quot;:&#44;&quot;key&quot;:&#58;&quot;CSRF&quot;:&#44;&quot;value&quot;:&#58;&quot;CSRF&quot;:&#125;}" />
<input type="submit" value="Submit request" />
</form>
</body>
</html>

```

Another malicious abuse consists into forcing a change password using an attacker-controlled value. See bug #3.

The application must use anti-CSRF arbitrary tokens to prevent those attacks. This is a standard practice in modern web applications.

## #3 Change password does not require old password - MEDIUM RISK

As mentioned in #2, Unifi Controller does not require the old admin password while changing credentials. This is an insecure design that can be easily abused by malicious users. Any session hijacking vulnerability or CSRF could result in full compromise.

#### #4 Frameable response (ClickJacking) - MEDIUM RISK

It might be possible for a web page controlled by an attacker to load the content Unifi web controller within an iframe on the attacker's page. This may enable a "clickjacking" attack (<https://www.owasp.org/index.php/Clickjacking>), in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

To effectively prevent framing attacks, the application should return a response header with the name **X-Frame-Options** and the value **DENY** to prevent framing altogether, or the value **SAMEORIGIN** to allow framing only by pages on the same origin as the response itself.

#### #5 Credentials are saved in plain-text within MongoDB - MEDIUM RISK

Administration credentials are stored in plain-text (within *ace*, *db.admin.find()*) and displayed in clear-text within the Unifi Controller web interface. From the security standpoint, this is a bad practice; many types of vulnerability, such as weaknesses in session handling, broken access controls, and cross-site scripting, would enable an attacker to leverage this behavior to retrieve the passwords of other application users.

Considering that the same credential is used by all APs SSH, this departure from best practice allows to compromise the entire platform.

#### #6 Multiple Cross-Site Scripting vulnerabilities (Stored and Reflected) in /api/, abusing Internet Explorer content sniffing - LOW RISK

Multiple /api/ endpoints allow to inject arbitrary HTML tags, as illustrated in the example below

- /api/s/default/get/setting
- /api/s/default/set/setting/connectivity
- /api/s/default/set/setting/country
- /api/s/default/set/setting/guest\_access
- /api/s/default/set/setting/mgmt
- /api/s/default/set/setting/rsyslogd
- /api/s/default/set/setting/snmp

- /api/s/test/set/setting/guest\_access

## Request

```
POST /api/s/test/set/setting/guest_access1ce7d<a>e17125d0164 HTTP/1.1
Host: 192.168.0.205:8443
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:21.0) Gecko/20100101 Firefox/21.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Referer: https://192.168.0.205:8443/manage/s/test
Content-Length: 1182
Cookie: maintab.pagesize=30; unifises=79b649f5195fdba6b26bb2e9d27aabc2
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache

json=%7B%22portal_enabled%22%3Atrue%2C%22auth%22%3A%22none%22%2C%22x_password%22%3A%22%22%2C%22redirect_enabled%22%3Afalse%2C%22redirect_url%22%3A%22%22%2C%22custom_ip%22%3A%22%22%2C%22expire%22%3A%22
...[SNIP]...
```

## Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/json;charset=ISO-8859-1
Content-Length: 1123
Date: Thu, 29 Aug 2013 14:24:05 GMT

{"data": [{"_id": "521f5985e4b03052d0bd8571", "allowed_subnet_1": null, "auth": "none", "authorize_use_sandbox": false, "custom_ip": "", "expire": "480", "gateway": "paypal", "key": "guest_access1ce7d<a>e17125d0164", "merchantwarrior_use_sandbox": false, "payment_enabled": false, "paypal_use_sandbox": false, "portal_customized": false, "portal_enabled": true, "portal_hostname": "", "portal_use_hostn
...[SNIP]...
```

The user-supplied `<a>` tag is included within the response body; This behavior demonstrates that it is possible to inject new HTML tags into the returned document.

This behavior can be abused by an attacker to perform Cross-Site Scripting against Internet Explorer users. JSON responses use the content-type *application/json*; the problem is that the default mime type list of Internet Explorer does not include that mime-type, thus it is possible to force the browser to sniff the content and display the page as HTML.

For all technical details, please refer to <http://blog.watchfire.com/wfblog/2011/10/json-based-xss-exploitation.html> The author covers in great detail a possible technique.

To prevent Content-Type sniffing in Internet Explorer and mitigate this attack, the application must include the following HTTP header in all HTTP responses:

**X-Content-Type-Options: nosniff**

Please refer to [http://msdn.microsoft.com/en-us/library/ie/gg622941\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ie/gg622941(v=vs.85).aspx)