Non-Obvious (Crypto) Bugs by Example

Non-Obvious (Crypto) Bugs by Example - Gregor Kopf, Publisher not stated.

- Published: date not stated
- Original: <https://web.archive.org/web/20110102233032/http://gregorkopf.de/slides_berlinsides_2010.pdf>
- Preserved from: https://web.archive.org/web/20110102233032/http://gregorkopf.de/slides_berlinsides_2010.pdf (manual-import) on 2026-08-07
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Introduction Showcase One Showcase Two The End

Non-Obvious Bugs by Example

Gregor Kopf

BerlinSides 2010

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

What and why?

Non-obvious (crypto) bugs As an example: two well-known CMS Easy to make, hard to spot Interesting to exploit Fun ;)

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

How?

The process from discovery to exploitation will be shown The code part that raised suspicion Observations and initial thoughts about the code Further analysis (technical background of the bug) Exploitation

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Let’s get started: Typo3

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

What Are We Looking at?

Typo3 will allow us to view (almost) arbitrary files Just use a URL like http://foobar/index.php?jumpurl=target.txt&locationData=1::1& juSecure=1&juHash=31337f0023 You need to supply a hash value juHash, which typo3 verifies before file access is granted Let's look at the code!

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Code

1 $hArr = a r r a y (2 $ t h i s −>j u m p u r l , 3 t 3 l i b d i v : : GP (' l o c a t i o n D a t a ') , 4 t 3 l i b d i v : : GP (' mimeType ') , 5 $ t h i s −>TYPO3 CONF VARS [' SYS '] [' e n c r y p t i o n K e y '] 6); 7 $ c a l c J u H a s h= t 3 l i b d i v : : shortMD5 (s e r i a l i z e ($ h A r r)) ; 8 $ j u H a s h = t 3 l i b d i v : : GP (' j u H a s h ') ; 9 i f ($ j u H a s h == $ c a l c J u H a s h) { 10 i f ($ t h i s −>l o c D a t a C h e c k ($ l o c a t i o n D a t a)) { 11 $ t h i s −>j u m p u r l = r a w u r l d e c o d e ($ t h i s −>j u m p u r l) ; 12 i f (t 3 l i b d i v : : v e r i f y F i l e n a m e A g a i n s t D e n y P a t t e r n ($ t h i s −>j u m p u r l) 13 && basename (d i r n a m e ($ t h i s −>j u m p u r l)) !== ' t y p o 3 c o n f ') { 14 i f (@ i s f i l e ($ t h i s −>j u m p u r l)) { 15 [...] 16 r e a d f i l e ($ t h i s −>j u m p u r l) ; 17 exit ; 18 }

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Observations

1 $hArr = a r r a y (2 $ t h i s −>j u m p u r l , 3 t 3 l i b d i v : : GP (' l o c a t i o n D a t a ') , 4 t 3 l i b d i v : : GP (' mimeType ') , 5 $ t h i s −>TYPO3 CONF VARS [' SYS '] [' e n c r y p t i o n K e y '] 6); 7 $ c a l c J u H a s h= t 3 l i b d i v : : shortMD5 (s e r i a l i z e ($ h A r r)) ;

To calculate juHash, a variable named encryptionKey is used encryptionKey is unknown to us, so we cannot supply the correct hash value. Or can we? Side note: juSecure is basically a MAC of jumpurl. It's built improperly, as encryptionKey is just appended at the end of the data.

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

shortMD5

What does shortMD5() do? 1 p u b l i c s t a t i c f u n c t i o n shortMD5 ($ i n p u t , $ l e n =10) { 2 r e t u r n s u b s t r (md5($ i n p u t) , 0 , $ l e n) ; 3 }

shortMD5() returns the first 5 bytes (10 hex chars) of the MD5 hash of its input Shortening hash values is generally OK, but 5 bytes is not quite much...

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Equals Operator in PHP

The supplied hash is compared with the computed hash using the PHP operator == That looks reasonable. However, the == operator has some issues In PHP, == is not typesafe PHP might perform nasty typecasting before the actual comparison is performed!

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

More on ==

From the PHP manual: 1 var dump (0 == " a ") ; // 0 == 0 −> t r u e 2 var dump (" 1 " == " 01 ") ; // 1 == 1 −> t r u e 3 var dump (" 10 " == " 1 e1 ") ; // 10 == 10 −> t r u e 4 var dump (1 0 0 == " 1 e2 ") ; // 100 == 100 −> t r u e

Uh, WTF? In PHP, 100 is equal to 1e2 when using the == operator.. Nice to know ;) Side node: scientific notation 1.234e2 = 1.234 · 102 = 123.4

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Idea

What if the computed hash looks like 0e66631337? The comparison operator will treat it as equal to 0 (0e66631337 = 0 · 1066631337 = 0). If we could influence the computed hash to take the desired form, then we'd know it would be equal to 0, which we could easily submit as our juHash value

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Thoughts on the Feasibility

The computed hash can be easily influenced, as jumpurl does not need to be canonical (e.g. we can just append ./ to the file name) But what's the probability of hitting one of the hash values we want? Let's assume the first byte has to be 0x0e. The following nibbles would then need to be numerical (i.e. only from 0 to 9) Let's further assume MD5 generates a random distribution. There are 16 values for each nibble (0 - f). Ten of them (0-9) are OK for us. We therefore have a chance of 10 5 16 = 8 that a nibble is numeric.

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Thoughts on the Feasibility

As all the nibbles are (assumed to be) independent, the 8 1 5 overall chance for a good hash is · 256 |{z} 8 | {z } first byte 8 left nibbles

That is about 0.009095 . . . %. In other words, in average we need 5498 tries before we hit a good hash value That's not terribly much.. Actually we need even less tries, as hashes like 000e1337 . . . are also OK.

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Attack

It's straight forward. Submit multiple requests for the same file For each request, prepend a ./ to the filename Always submit 0 as juHash value Get some beerˆW coffee and wait for your file

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

For your Amusement

The code should actually check that you don't download localconf.php, which contains encryptionKey In fact, if MAGICQUOTES GPC is disabled, it doesn't Just use a file name like typo3conf/localconf.php%00/foobar/aa Once you got the encryption key, you can calculate the correct juHash value for any file you like

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Demo!

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Even more fun: Joomla

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Code

Looking through the code, one stubles upon the function genRandomPassword(). Interesting :) 1 f u n c t i o n genRandomPassword ($ l e n g t h = 8) { 2 $ s a l t = ” abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 ” ; 3 $len = strlen ($salt); 4 $makepas s = ’ ’ ; 5 6 $stat = @stat (FILE) ; 7 i f (empty ($ s t a t) | | ! i s a r r a y ($ s t a t)) $ s t a t = a r r a y (php uname ( )) ; 8 9 mt srand (crc32 (microtime ( ) . implode (’ | ’ , $ s t a t))) ; 10 11 f o r ($ i = 0 ; $ i < $ l e n g t h ; $ i ++) { 12 $makepas s .= $ s a l t [mt rand (0 , $ l e n −1)]; 13 } 14 15 r e t u r n $makepas s ; 16 }

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Observations

The used PRNG is the Mersenne Twister (seeded with 32 bit values) For each length, there are at most 232 passwords Reseeding the PRNG for every password is not exactly smart The seed is obtained using CRC32 CRC input values are the system time and the output of stat() The only things that change in the CRC input are the time fields CRC32 is not a cryptographic hash! Maybe the seed is predictable?

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Impact?

Even if we could predict the seed: what would it be good for? The affected function is used for generating password reset tokens: 1 // G e n e r a t e a new t o k e n 2 $ t o k e n = J U t i l i t y : : g e t H a s h (J U s e r H e l p e r : : genRandomPassword ( )) ; 3 $ s a l t = J U s e r H e l p e r : : g e t S a l t (’ c r y p t −md5 ’) ; 4 $has hedToken = md5($ t o k e n . $ s a l t) . ’ : ’ . $ s a l t ; 5 6 $query = ’UPDATE # u s e r s ’ 7 . ’ SET a c t i v a t i o n = ’ . $db−>Quote ($has hedToken) 8 . ’ WHERE i d = ’ . (i n t) $ i d 9 . ’ AND b l o c k = 0 ’ ; 10 11 $db−>s e t Q u e r y ($ q u e r y) ;

Password reset admin account fun/profit

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

getHash

To generate a password reset token, the function getHash() is used: 1 f u n c t i o n getHash ($seed) 2 { 3 $ c o n f =& J F a c t o r y : : g e t C o n f i g ( ) ; 4 r e t u r n md5($ c o n f−>g e t V a l u e (’ c o n f i g . s e c r e t ’) . $seed); 5 }

config.secret is a random string generated during the installation process genRandomPassword() is used to generate config.secret 1 $vars [’ s i t e U r l ’] = JURI : : r o o t ( ) ; 2 $vars [’ secret ’] = J U s e r H e l p e r : : genRandomPassword (1 6) ; 3 4 $vars [’ o f f l i n e ’] = J T e x t : : (’STDOFFLINEMSG ’) ;

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Short Summary

The password reset function generates a reset token and sends it out via e-mail Uses a randomly generated string Also uses an installation-specific secret key :(We need to find a way to predict the randomly generated string We also need to know the secret key Looks challenging. Let’s go!

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

How to Obtain config.secret

config.secret is used in a number of places Whenever you click ,,remember my password", a cookie will be set. The cookie's name is determinded by the following code: 1 $ c r y p t = new J S i m p l e C r y p t ($ k e y) ; 2 $ r c o o k i e = $ c r y p t−>e n c r y p t (s e r i a l i z e ($ c r e d e n t i a l s)) ; 3 $ l i f e t i m e = t i m e ( ) + 365 24 60 60; 4 s e t c o o k i e (J U t i l i t y : : g e t H a s h ('JLOGIN REMEMBER ') , $ r c o o k i e , 5 $lifetime , '/ ');

getHash() is used here again, so cookie = md5(config .secret + JLOGIN REMEMBER)

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

How to Obtain config.secret

config.secret is generated during the installation process using the password generation function we have already seen There are only 232 possible passwords, so we could build a table to look up the used seed based on the observed authentication cookie name That costs us 232 memory and 232 time Could be optimized using rainbow tables It's a great stress test and benchmark for your hardware ;)

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Next Steps

Alright, we can get config.secret. What now? We would like to predict the seed is was used to initialize the PRNG when we reset some password CRC is used to generate that seed. Let's have a closer look at CRC Cyclic Redundancy Check Based on polynomials over F2

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

More CRC

Message m is interpreted as a polynomial over F2 , taking the bits as coefficients (MSB x 0) CRC (m) := x N · poly(m) mod g for some fixed polynomal g (one can say that CRC operates on a polynomal ring) The multiplication by x N is for technical reasons. For CRC32: N = 32 Example: 11001b 1 · x 0 + 1 · x 1 + 0 · x 2 + 0 · x 3 + 1 · x 4 The message polynomal is divided by a fixed generator polynomial (polynomial division, you might remember it from school) The remainder is the CRC value

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

So?

An interesting property: CRC is additive! CRC (m) + CRC (n) = CRC (m + n) Addition is of course in F2 I.e. poly(m) + poly(n) = poly(m xor n)

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

So?

To put it in other words Assume we have some message m but we only know its CRC value c We can now generate CRC values CRC (m xor n), where n is another message That means: we can selectively change bits in the message and (without even knowing the message!) obtain according

CRC values Once we know one CRC value used for PRNG initialization, we could try to use it to predict future CRC values

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Idea

Reset our own password and obtain a token Use the token to obtain the CRC value that was used to initialize the PRNG Again, there are only 232 possibilities The CRC value can be guessed or a (site specific, as token depends on config.secret) table can be build Use the obtained CRC value to calculate future CRC values Reset the password of the admin account and guess the token

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Flipping the Bits

The input to the CRC function was 1 crc32 (microtime ( ) . implode (' | ' , $ s t a t))

Between two calls, only the first few bits in the CRC argument change More precisely, as microtime() is used in a string context, only the lower nibbles of the first few bytes can change (e.g. from 0x30 to 0x33 or so)

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Flipping the Bits Sample output of microtime(): |0.95003500 {z } 1283184410 | {z } fraction of seconds system time The last two bytes of the first part are often zero If we manage to issue two password reset requests within the same 10ms, then the potentially flipped bits are represented by the following mask: 0x 00000000 | {z } 0f | 0f{z0f 0f} 00000 | {z . .}. 0.XX not changed potentially flipped low nibbles last part not changed So let's just compute the CRC of those flipped bits and add it to the CRC we already know from our token! Erm, wait. How many zeroes are there at the end? We also need to know the length of the CRC input string Unfortunately, that depends on the output of stat(), which we cannot predict Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Finding the Original Input Length

We can generate two reset tokens for our own account We know that the input to the CRC function only differs in a few bits XORing the two CRC values results in the CRC value d of the bit difference of both original inputs Both CRC inputs have an unknown length l The bit difference must have the form 1011001 | {z . .}. 0000000 | {z . .}. k bits that make the difference l−k zero bits

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Finding the Original Input Length

Now it gets interesting ;) Say we have the CRC d of the bit difference m and we want to find the original input length l We know the bit difference has the form m · X l , i.e. only the first few bits may have changed The equation we want to solve looks like this: X 32 · m · X l g d Keep in mind: X , m and d are polynomials, x g y is shorthand for x = y mod g

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Finding the Original Input Length

Lucky us, in case of CRC32 g is irreducible, i.e. X 32+l is invertible We can use the extended version of euclids algorithm to compute (X 32+l)−1 , where (X 32+l)−1 · X 32+l = 1 That gives us m g d · (X

32+l)−1 If we assume m < g , then obviously m mod g = m. In that case we can therefore simply write m = d · (X 32+l)−1 Although we neither know m nor l, we can still enumerate different values for l and see if one of the resulting m will match our constraints regarding the flipped bits (only the lower nibbles are flipped) That will typically give us one or two candidates for l. Iterate the process to determine l Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Full Attack

Log in on the target site and click ,,remember my password" Use the obtained cookie name to look up the value config.secret Reset your own password a couple of times Reset the password of the admin account Use the obtained tokens to get the CRC32 values that were used to initialize the PRNG Use a pre-calculated (application specific!) table Or perform a live brute force search

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

The Full Attack

Use the obtained CRC32 values to calculate the length l of the input to the CRC32 function Now enumerate all possible bit differences (e.g. 0x |000000000f 0f{z0f 0f 00000 . .}.), and compute their CRCs l bytes Add the computed CRCs to the CRC that was used to initialize the PRNG for your own token Use the obtained CRCs to initialize the PRNG and to build tokens based on config.secret and a randomly generated string Get some beerˆW vodka and wait until you hit the right token

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Demo!

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Conclusions (Typo3)

What went wrong? Shortening a MAC value without proper reasons We have enough bandwidth to submit full hash values ;) Using a not-typesafe comparison operator Further: forgetting about null bytes

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Conclusions (Joomla)

Using a weak PRNG 32 bit seed No entropy accumulator Frequently reseeding the PRNG Using CRC32 for cryptographic purposes

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Contact me: mail: ping@gregorkopf.de twitter: teh gerg

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Sploit demo: Typo3

[greg@uchuck ~/research/typo3]$ python sploit.py http://127.0.0.1/t3/index.php /../../../../../etc/passwd [.] Done 100 tries. [+] Success after 142 tries! [+] Download link: http://127.0.0.1/t3/index.php?jumpurl=/../../../../../etc/passwd &locationData=1::3541&juSecure=1&juHash=0 # $FreeBSD: src/etc/master.passwd,v 1.40.22.1.4.1 2010/06/14 02:09:06 kensmith Exp $ root::*0:0:Charlie &:/root:/bin/csh toor*::0:0:Bourne-again

Superuser:/root: daemon:*:1:1:Owner of many system processes:/root:/usr/sbin/nologin operator:*:2:5:System &:/:/usr/sbin/nologin [...]

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Sploit demo: Joomla [greg@uchuck ~/research/joomla]$ python feierAndForget.py 127.0.0.1 /joomla 'greg@uchuck' 'root@uchuck' [+] Getting cookie.. [+] Cookie = 0adafef00f88ef16c63573cbc80ec425= ade360257773f5c36186bfa4489d57c6 [+] Got remember cookie: 8dfa83e4cf5cae043b797a3c2a9fdee4 [+] Looking it up in the tables: ..... [+] CRC value 0xD5E47F7E was used to generate config.secret [+] config.secret = OYHDHQbgoYMSETeT [+] Precomputed tables found! Going on. [+] Establishing new session.. [+] Reset requests sent. Check your mail! [.] Please enter token 1: c13308a6e411f270ce39b4a80d4ca591 [.] Please enter token 2: 241a5822289bae9bfa8cc28ba2a425f3 [+] Thanks. Now looking up token1 in the specific tables: ..... [+] Found token1. CRC = 0x129FACBB [+] Now for token2: ..... [+] Found token2. CRC = 0x24083D4E [+] CRC pre-image length: 176 [+] Please, only one more token: 0daacacae08c6956394aeb94d5d67094 .....

Non-Obvious Bugs by Example Introduction Showcase One Showcase Two The End

Sploit demo: Joomla (contd.)

[X] Time to try: python bruteToken.py OYHDHQbgoYMSETeT 0x96855789 000000000f0f0f0f\ 176 127.0.0.1 /joomla admin

[greg@uchuck ~/research/joomla]$ python bruteToken.py\ OYHDHQbgoYMSETeT 0x96855789\ 000000000f0f0f0f 176 127.0.0.1\ /joomla admin [+] Getting cookie.. [+] Cookie = 0adafef00f88ef16c63573cbc80ec425= 4f891a91a889f5595671122582e50fbe [+] Got hidden field: 51e2a6f3a5e0d324930875ae97bb98b3 ..... Your username / token: admin / b3b729db0688260e12da1c32e6375231 URL http://127.0.0.1//joomla/index.php?option=com_user&view=reset&layout=confirm

Non-Obvious Bugs by Example