Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Pentest-Report NTPsec 01.2017

Cure53, Dr.-Ing. M. Heiderich, M. Wege, MSc. N. Krein, BSc. D. Weißer

Index

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Introduction

“Welcome to the NTPsec project - a secure, hardened, and improved implementation of Network Time Protocol derived from NTP Classic, Dave Mills’s original.

NTPsec, as its name implies, is a more secure NTP. Our goal is to deliver code that can be used with confidence in deployments with the most stringent security, availability, and assurance requirements.

Towards that end we apply best practices and state-of-the art technology in code auditing, verification, and testing. We begin with the most important best practice: true open-source code review. The NTPsec code is available in a public git repository. One of our goals is to support broader community participation.”

From https://www.ntpsec.org/

This report documents the findings of a source code audit of the NTPsec software. The project was completed by Cure53 team in January 2017. Four members of the Cure53 participated in this assignment, which required a total of thirty-two days of testing in order for a satisfactory level of coverage to be reached.

The audit constituted a joint project dedicated to both the NTPsec and the NTP. The code base of the NTP was examined in parallel and the two components of the scope were given the same amount of attention and scrutiny. While this document primarily pertains to the NTPsec element, the relevant results applicable to NTP are also briefly recalled. At the same time, a separate report has been created to discuss the NTP issues in detail and at length. In the latter NTP-related document, the discoveries connected to NTPsec are analogically given less space and specificity in reporting.

As for the test’s approach, the investigations were rooted in the so-called white-box methodology, meaning that the testing team was granted full access to relevant sources and the like. Prior to initiating the audit, the Cure53 team established solid communication channels for the two respective software items in scope, liaising with the development teams of the NTPsec and NTP, respectively.

The document proceeds with describing the test’s scope, then discusses coverage and findings, ultimately delivering conclusions and verdicts about the general level of security discovered, and the state of the audited code for the two software products in question.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Scope

- **NTPsec 0.9.6**
  - https://ftp.ntpsec.org/pub/releases/ntpsec-0.9.6.tar.gz

Test Coverage

One can consult an overview of the test's coverage below. The listing shows the percentage of the coverage reached by the code audit per directory. All directories belong to the downloaded code base.

Coverage in %	SLOC	Directory
100	38094	ntpd
20	6175	libntp
n/a	5933	tests
100	3955	include
n/a	3884	ntpclients
n/a	3745	libparse
40	2822	libisc
n/a	2220	wafhelpers
n/a	1679	pylib
n/a	950	libjsmn
100	483	ntpfrob
100	471	libsodium
100	435	ntptime
100	416	attic
n/a	130	devel
n/a	111	contrib

The directories marked with *"n/a*" were explicitly left out of scope along with all of the subcomponents written in Python and the testing framework unity.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following sections list both vulnerabilities and implementation issues spotted during the testing period. Note that findings are listed in a chronological order rather than by their degree of severity and impact. The aforementioned severity rank is simply given in brackets following the title heading for each vulnerability. Each vulnerability is additionally given a unique identifier (e.g. *NTP-01-001*) for the purpose of facilitating any future follow-up correspondence.

NTP-01-002 NTP: Buffer Overflow in *ntpq* when fetching *reslist* (*Critical*)

Note: This issue affects NTP only and is not present in the NTPsec code.

A stack buffer overflow can be triggered by a malicious server when a client (using *ntpq*) requests the restriction list from the server. This is due to a missing *length* check in the *reslist()* function. It occurs whenever the function parses the server's response and encounters a *flagstr* variable of an extensive *length*. The string will be copied into a fixed-size buffer, leading to an overflow on the function's stack-frame.

NTP-01-012 NTPsec: Authenticated DoS via Malicious Config Option (*High*)

Note: This issue affects both NTP and NTPsec and is present in both code bases.

A vulnerability found in the NTPsec server allows an authenticated remote attacker to crash the daemon by sending an invalid setting via the *:config* function. The *unpeer* option expects either a number or an address as argument. In case the value is "*0*", a segmentation fault occurs. An example is given in the following listing.

Configurating the server remotely:

```
ntpq> :config unpeer 0
Keyid: 1
MD5 Password:
localhost: timed out, nothing received
***Request timed out
```

The submission of the configuration crashes the NTPsec server right away. An observation performed with the GDB demonstrates that the error occurs in *ntp_config.c* and is due to a *null* pointer dereference.

Segmentation fault:

```
(gdb) r
[...]
Program received signal SIGSEGV, Segmentation fault.
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
0x0000000000407c1e in config_unpeers (ptree=0x65c0b0)
at ../../ntpd/ntp_config.c:3042
3042                    AF(&peeraddr) = curr_unpeer->addr->type;
(gdb) p *curr_unpeer
$1 = {link = 0x0, assocID = 0, addr = 0x0}
```

The *unpeer* configuration options are processed in the *config_unpeers()* function in *ntpd/ntp_config.c*. The *curr_unpeer* struct contains the provided parameter which is either a number or an address. While *assocID* holds numeric values, the *addr* is a pointer to another struct for when the parameter is an address. If the *curr_unpeer->assocID* is zero, then the code expects *curr_unpeer->addr.* However, this is not necessarily the case. Setting "*unpeer 0*" leads to a completely empty *curr_unpeer* struct and thereby crashes the server.

Affected File:
ntpsec/ntpd/ntp_config.c

Affected Code:

```
config_unpeers(
       config_tree *ptree
       )
{
[...]
       curr_unpeer = HEAD_PFIFO(ptree->unpeers);
       for (; curr_unpeer != NULL; curr_unpeer = curr_unpeer->link) {
[...]
              if (curr_unpeer->assocID) {
[...]
                     continue;
              }

              ZERO(peeraddr);
              AF(&peeraddr) = curr_unpeer->addr->type;
```

Verifying the *addr* element in case the *assocID* is zero is a way towards mitigating this issue properly.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

NTP-01-015 NTPsec: Regression in *ctl_putdata()* leads to Endless Loop *(High)*

Note: This issue affects NTPsec only and is a regression from a security fix.

Part of this assignment related to the verification process regarding the implementation of fixes to the formerly reported problems proven to affect NTP. Checking how NTPsec handles these matters revealed a highly important security patch suffering from regression. More specifically, the original vulnerability tracked under *CVE-2014-9295* was fixed by an initial patch[1] on the 12th of December, 2014. However, the code in question appears to have undergone changes and revisions on or around November 4th, 2016, which removed the fix. These can be seen in a subsequent *commit*[2] from that period. Strangely enough, the rewrite removes nearly the entire patch introduced to prevent the original vulnerability. As a consequence, the code now introduces an authenticated DoS. The affected code is displayed below.

Affected File:
ntpsec/ntpd/ntp_control.c

Affected Code:

```
static void
ctl_putdata(
const char *dp,
       unsigned int dlen,
       bool bin            /* set to true when data is binary */
       )
{
[...]

       while (dlen + overhead + datapt > dataend) {
              ctl_flushpkt(CTL_MORE);
       }

       memcpy(datapt, dp, dlen);
       datapt += dlen;
       datalinelen += dlen;
       datasent = true;
}
```

The *while*-loop above runs endlessly in the current setup as long as *dlen* is so large that, even when *datapt* points to the beginning of the buffer, the addition of *dlen + overhead + datapt* points outside of the *dataend*. Given that *ctl_flushpkt()* only resets *datapt* and the

[1] https://github.com/ntpsec/ntpsec/commit/7fd82020dfd501ee4510edbd61eaf1eb796d5db9
[2] https://github.com/ntpsec/ntpsec/commit/1a545205529b17390a7ae93bfc069b5a517c95bc

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

decrementation of *dlen* has been removed from the code, this condition can never amount to *false* once the loop is entered for the initial time. This can be verified by modifying the *ntpq* code to allow sending longer variables and receiving them back:

```
$ ./ntpq
ntpq> :config setvar a = 
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAA
Keyid: 1
MD5 Password:
Config Succeeded
ntpq> rv 0 a
localhost: timed out, nothing received
***Request timed out
ntpq>
```

Because NTPsec resembles NTP in that it *forks* only during the startup process, this issue signifies a permanent DoS where *ntpd* is required to be restarted to function properly again. It is recommended to review the regression and possibly reintroduce the fix to completely and consistently mitigate all issues resulting from overly long outgoing packets.

NTP-01-016 NTPsec: Denial of Service via Malformed Config *(High)*

Note: This issue affects both NTP and NTPsec and is present in both code bases.

A vulnerability found in the NTPsec server makes it possible for an authenticated remote user to crash the service via a malformed configuration. After submitting the *config* line in the following snippet, the *ntp daemon* crashes after a couple of seconds.

Configuring the server remotely:

```
ntpq> :config server 10.0.0.1 mode 3735928559
Keyid: 1
MD5 Password:
Config Succeeded
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

The exact reason for the crash can be derived from running the daemon in the GDB. Here an invalid value in the *%rax* register leads to an invalid *read* operation and causes a segmentation fault.

Segmentation fault:

```
(gdb) r
[...]
Program received signal SIGSEGV, Segmentation fault.
0x0000000000415f1f in peer_xmit (peer=0x654980 <init_peer_alloc+704>) at
../../ntpd/ntp_proto.c:2158
2158                sendpkt(&peer->srcadr, peer->dstadr, sys_ttl[peer->ttl],
                            &xpkt, sendlen);
(gdb) x/i $rip
=> 0x415f1f <peer_xmit+480>:     movzbl 0x658460(%rax),%eax
(gdb) p/x $rax
$1 = 0xdeadbeef
(gdb) p/x peer->ttl
$2 = 0xdeadbeef
```

The affected code is in the *ntp_proto.c* file of the *peer_xmit()* function. *Peer* is a struct which contains several values, including the user-controlled *peer->ttl* variable. An invalid value causes an invalid memory access.

Affected File:
ntpsec/ntpd/ntp_proto.c

Affected Code:

```
peer_xmit(
       struct peer *peer   /* peer structure pointer */
       )
{
[...]
       sendpkt(&peer->srcadr, peer->dstadr, sys_ttl[peer->ttl],
              &xpkt, sendlen);
```

The core problem resides within the configuration parser where the parameters from the configuration lines are stored in node structs. Setting the *ttl* value can be done in three different ways (“*ttl*”, “*subtype”*, and “*mode*”) but only one of them investigates the provided number for sanity. While invalid numbers are being ignored when “*ttl*” option is used, no checks are performed for “*mode*”.

Affected File:
ntpsec/ntpd/ntp_config.c

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
peer_node *
create_peer_node(
       int          hmode,
       address_node *      addr,
       attr_val_fifo *     options
       )
{
[...]
       case T_Ttl:
              if (option->value.u >= MAX_TTL) {
                     msyslog(LOG_ERR, "ttl: invalid argument");
                     errflag = true;
              } else {
                     my_node->ctl.ttl = (uint8_t)option->value.u;
              }
              break;

       case T_Subtype:
       case T_Mode:
              my_node->ctl.ttl = option->value.u;
              Break;
```

It was not possible to exploit this issue beyond achieving DoS. However, it is recommended to resolve this problem by adding sanity checks to the *subtype* and *mode* configuration options.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers those noteworthy findings that did not lead to an exploit but might aid an attacker in achieving their malicious goals in the future. Most of these results are vulnerable code snippets that did not provide an easy way to be called. Conclusively, while a vulnerability is present, an exploit might not always be possible.

NTP-01-001 NTPsec: Makefile does not enforce Security Flags *(Low)*

Note: This issue affects both NTP and NTPsec and is present in both code bases.

One of the key realms reviewed quite early during almost every security test of a new project encompasses studying the presence of hardening flags applied when the software is built. This can be done with tools like *checksec*[3] or *PEDA*[4] once the software has been compiled with the default options inside the *makefile* at hand:

```
$ gdb ./ntpd
Reading symbols from ./ntpd...done.
(gdb) checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : ENABLED
RELRO     : Partial
(gdb)
```

From the GDB's output it is apparent that the hardening flags are derived from the global Linux distribution setting rather than forced from the *makefile* itself. From this follows that certain hardening checks are missing. These include stack canaries, which ordinarily protect the *return address* from buffer overflow vulnerabilities on the stack, as well as *FORTIFY_SOURCE.*

It is important to set the necessary *CFLAGS* inside the *makefile* itself in order to directly instruct the compiler to insert all of the security flags required. Once activated, the exploitation of multiple kinds of memory corruption vulnerabilities becomes much more difficult. This increase in security stems from two reasons: one having to do with requiring additional information leaks from the program's memory, and the other revolving around establishing that the problems are mitigated by, for example, newly introduced *length* checks.

[3] http://www.trapkit.de/tools/checksec.html
[4] https://github.com/longld/peda

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

The following snippet shows what *CFLAGS* are recommended for an addition to the *make* process:

```
$ make CFLAGS='-Wl,-z,relro,-z,now -pie -fPIE -fstack-protector-all
-D_FORTIFY_SOURCE=2 -O1'
[...]
$ gdb ./ntpd
Reading symbols from ./ntpd...done.
(gdb) checksec
CANARY    : ENABLED
FORTIFY   : ENABLED
NX        : ENABLED
PIE       : ENABLED
RELRO     : FULL
(gdb)
```

NTP-01-003 NTPsec: Improper use of *snprintf()* in *mx4200_send()* *(Low)*

Note: This issue affects both NTP and NTPsec and is present in both code bases.

The function *mx4200_send()* uses the *libc* function *snprintf()/vsnprintf()* incorrectly. This can lead to an out-of-bounds memory write due to an improper handling of the return value of *snprintf()/vsnprintf()*. Said value returns the number of bytes it would have written if there were no length restrictions in place.

The code in question takes the return value outlined above and increments an iterator by its value. This iterator is supposed to point into the fixed-size buffer. However, since the return value can be larger than the buffer's size, it is possible for the iterator to point somewhere outside of the allocated buffer space. This results in an out-of-bound memory *write* in the *snprintf()* specified in this ticket. The reason behind the problem is that the iterator is used as the destination pointer.

This behavior can be leveraged to overwrite a saved instruction pointer on the stack and gain control over the execution flow. During the test it was not possible to identify any malicious usage for this function, specifically no way for an attacker to exploit the issue mentioned above was ultimately unveiled. However, this remains to be a problem capable of introducing new vulnerabilities. The problems are likely to resurface when new code that uses this function is added. In other words, it is necessary to fix this flaw in advance.

Affected File:
ntpsec/ntpd/refclock_magnavox.c

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
static void
mx4200_send(struct peer *peer, char *fmt, ...)
{
[...]
char buf[1024];
[...]

        cp = buf;
        *cp++ = '$';
        n = vsnprintf(cp, sizeof(buf) - 1, fmt, ap);
        ck = mx4200_cksum(cp, n);
        cp += n;
        ++n;
        n += snprintf(cp, sizeof(buf) - n - 5, "*%02X\r\n", ck);
```

In the above code, *cp* initially points to the beginning of the buffer. Once the *vsnprintf()* returns, *mx4200_cksum()* is called for creating a checksum. This is done by iterating over each byte of the buffer. However, the *mx4200_cksum()* uses the return value *n* from *vnsprintf()* to determine the length of the buffer it needs to iterate over. Since *n* may be larger than the buffer's size, an out-of-bounds read can occur as a result of creating the checksum.

It is recommended to check the return value of the *vsnprintf()/snprintf()* and ensure that it does not exceed the size allowed for a buffer. Also, before calling *snprintf()*, it must be ensured that at least five bytes are available, with the view to avoiding an overflow.

NTP-01-004 NTPsec: Potential Overflows in *ctl_put()* functions *(Medium)*

Note: This issue affects both NTP and NTPsec and is present in both code bases.

For the purpose of formatting different kinds of response strings into each response packet, *Ntpd* makes use of different wrappers around *ctl_putdata()*. For example, *ctl_putstr()* is often used to send quoted system variables, while *ctl_putuint()* comes into the fore when integer responses are being handled. All of these wrappers, however, suffer from stack based buffer overflow vulnerabilities as soon as they are utilized incorrectly. This is due to the fact that the length of the source variable is used on each occasion when the data is being copied into a local buffer. This is highlighted in the provided code.

Affected File:
ntpsec/ntpd/ntp_config.c

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Affected Code:

```
ctl_putstr(
       const char *  tag,
       const char *  data,
       size_t        len
       )
{
       char buffer[512];
       char *cp;
       size_t tl;

       tl = strlen(tag);
       memcpy(buffer, tag, tl);
```

While this issue should be considered hard to exploit with the presence of the stack canaries and is actually mitigated by *FORTIFY_SOURCE*, these functions nevertheless pose a considerable threat as soon as they operate on values larger than the destination size.

Although the current state of NTP appears not to permit setting tag lengths greater than 512 bytes (mainly because they all have static values), it is still recommended to fix all *ctl_put* functions by limiting the source length

NTP-01-005 NTPsec: Off-by-one in Oncore GPS Receiver *(Low)*

Note: This issue affects both NTP and NTPsec and is present in both code bases.

Regardless of bugs inside the *refclock* drivers not posing high security risks, the Cure53 testing team discovered several coding errors worth reporting. One mistake was found in the Oncore GPS Receiver of Motorola devices. The vulnerable code can be found below.

Affected File:
ntpsec/ntpd/refclock_oncore.c

Affected code:

```
static void
oncore_receive(
       struct recvbuf *rbufp
       )
{
       size_t i;
       u_char *p;
```

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

```
        struct peer *peer;
        struct instance *instance;

        peer = rbufp->recv_peer;
        instance = peer->procptr->unitptr;
        p = (u_char *) &rbufp->recv_space;
[...]
        i = rbufp->recv_length;
        if (rcvbuf+rcvptr+i > &rcvbuf[sizeof rcvbuf])
                i = sizeof(rcvbuf) - rcvptr;     /* and some char will be lost */
        memcpy(rcvbuf+rcvptr, p, i);
        rcvptr += i;
        oncore_consume(instance);
}
```

The highlighted length check above incorrectly sets the boundaries for the received buffer by limiting to *sizeof(rcvbuf).* In this context, an alternative *sizeof(rcvbuf) - 1* would be correct because the size is used as an index. This creates an *off-by-one* buffer overflow. Since *rcvbuł* is directly followed by another buffer, this issue is deemed nearly impossible to exploit. Still, it should be viewed as a coding error and resolved accordingly.

NTP-01-006 NTP: Copious amounts of Unused Code *(Info)*

Note: This issue affects NTP only and is not present in the NTPsec code.

Statically included external projects potentially introduce several problems and the issue of having extensive amounts of code that is “dead” in the resulting binary must clearly be pointed out. The unnecessary unused code may or may not contain bugs and, quite possibly, might be leveraged for code-gadget-based branch-flow redirection exploits.

NTP-01-007 NTP: Data Structure terminated insufficiently *(Low)*

Note: This issue affects NTP only and is not present in the NTPsec code.

Calling *strcpy()* with an argument of *string* with additional *null* bytes actually only copies a single terminating *null* character into the target buffer instead of relying on the required *double null* bytes in *addKeysToRegistry()* function. As a consequence, a garbage registry entry can be created and consist leaked memory contents.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

NTP-01-008 NTP: Stack Buffer Overflow from Command Line *(Low)*

Note: This issue affects NTP only and is not present in the NTPsec code.

Invoking *strcat()* blindly appends the passed string to stack buffer in the *addSourceToRegistry()* function. The stack buffer is 70 bytes smaller than the buffer in the calling *main()* function. Together with the initially copied *Registry* path, the combination causes a stack buffer overflow and effectively overwrites the stack frame.

NTP-01-009 NTP: Privileged execution of User Library code *(Low)*

Note: This issue affects NTP only and is not present in the NTPsec code.

The Windows NT port has the added capability to preload DLLs defined in the inherited global local environment variable *PPSAPI_DLLS.* The code contained within those libraries is then called from the NTPD service, usually running with elevated privileges.

NTP-01-010 NTP: *ereallocarray()/eallocarray()* underused *(Info)*

Note: This issue affects NTP only and is not present in the NTPsec code.

NTP makes use of several wrappers around the standard heap memory allocation functions that are provided by *libc*. This is mainly done to introduce additional safety checks concentrated on several goals. The described function additionally ensures that the later multiplication of *size* * *nmemb* does not create an integer overflow. In other words, it is responsible for attesting to less memory than originally intended not being allocated in a given case. The problem, however, is that the function in question is used quite rarely, even though there are some places calling for it to be employed instead of the usual *emalloc*.

NTP-01-011 NTP: *ntpq_stripquotes()* returns incorrect Value *(Low)*

Note: This issue affects NTP only and is not present in the NTPsec code.

The NTP client (*ntpq*) uses the function *ntpq_stripquotes()* to remove quotes and escape characters from a given string. According to the documentation, the function is supposed to return the number of copied bytes but due to incorrect pointer usage this value is always zero.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

NTP-01-013 NTPsec: Inclusion of obsolete NTPclassic-dependent Script *(Info)*

Note: This issue affects NTPsec only and is not present in the NTP code.

The NTPsec project includes an inapplicable script dependent on the NTPclassic's *ntpq*. This inclusion is believed to be a mere oversight, which can be likely attributed to the challenges of the repository conversion.

Affected File:
ntpsec/attic/ntpver

Affected Code:

```
ntpq -c "rv 0 daemon_version" $* | awk '/daemon_version/ { print $2 }'
```

It is recommend for the tool to be removed. An alternative dependency linked to the *NTPclassic* shall be added or the script is to be replaced with an implementation using *ntpdig*.

NTP-01-014 NTP: Buffer Overflow in DPTS Clock *(Low)*

Note: This issue affects NTP only and is not present in the NTPsec code.

Another potential issue inside the *refclock* drivers was found in the receiver for the Datum Programmable Time Server. Here the packets are processed from the */dev/datum* device and handled in *datum_pts_receive()*. Since *dpend* simply holds the length of the entire packet, the loop highlighted above will continue the process of copying data into *datum_pts->retbuf*, even though there is only room for 8 bytes there. This is a classic buffer overflow inside a data structure that is stored on the heap.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Conclusion

The joint nature of this January 2017 code audit, performed by the Cure53 team against both the NTPsec and the NTP software components in scope, makes it that much complex to issue an unambiguous verdict about all security-relevant aspects.

The bottom line is that four members of the Cure53 team, who assessed the products over the course or thirty-two days, discovered sixteen individual findings in the code base of both NTPsec and NTP. It should be noted, however, that the sole finding flagged with a “Critical” severity concerned NTP. Breaking down the findings moreover indicates that eight of the discoveries were exclusively tied to NTP entity, while a much smaller array of two issues could be linked exclusively to the realm of NTPsec. This means that six spotted problems were shared between the two code bases. In other words, the total numbers of findings suggest a slightly lower number of eight native problems for NTPsec compared to the fourteen issues affecting NTP.

The general outcome of this project is rooted in the fact that NTP’s code has been left to grow organically and had aged somewhat unattended over the years. The overall structure has thus become very intricate, while also yielding a conviction that different styles and approaches were used and subsequently altered. The seemingly uncontrolled inclusion of variant code via header files and complete external projects engenders a particular problem. Most likely, it makes the continuous development much more difficult than necessary. While the NTPsec project emphasizes cleaning up its ancestors’ flaws, the difference regarding quality between the original code and the current implementation was not as great as anticipated. The more recent project suffers from having convoluted code and allowing for obsolete nooks and crannies to persist in the code base.

On the one hand, much cruft has been removed successfully, yet, on the other hand, the code shared between the two software projects bears tremendous similarities. The NTPsec project is still relatively young and a major release has not yet occurred, so the expectations are high for much more being done beforehand in terms of improvements. It must be mentioned, however, that the regression bug described in NTP-01-015 is particularly worrisome and raises concerns about the quality of the actions undertaken.

In sum, one can clearly discern the direction of the project and the pinpoint the maintainers’ focus on simplifying and streamlining the code base. While the state of security is evidently not optimal, there is a definite room for growth, code stability and overall security improvement as long as more time and efforts are invested into the matter prior to the official release of NTPsec.

Dr.-Ing. Mario Heiderich, Cure53
Rudolf Reusch Str. 33
D 10367 Berlin
cure53.de · mario@cure53.de

Cure53 would like to thank Gervase Markham of Mozilla for his excellent project coordination, support and assistance, both before and during this assignment. Cure53 would further like to extend gratitude to the NTPsec team, for their help during the scoping phase of this assessment.