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 Dovecot 11.2016

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

Index

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

Introduction

"Dovecot is an open source IMAP and POP3 email server for Linux/UNIX-like systems, written with security primarily in mind. Dovecot is an excellent choice for both small and large installations. It's fast, simple to set up, requires no special administration and it uses very little memory."

From http://www.dovecot.org/

This report documents the findings of a penetration test and source code audit against the Dovecot email server software. The project was carried out by four testers of the Cure53 team over the course of twenty days in October and November 2016.

In the broader web and security community, Dovecot is known for being very much robust and secure[1]. With good reasons, the positive assessment of the software, which is unlike many others regarding fulfilment of its security promises, is largely used for promotional and advertising purposes. On the Index page of the Dovecot website, one can read that

> *"Dovecot's design and implementation is highly focused on security. Rather than taking the traditional road of just fixing vulnerabilities whenever someone happens to report them, I offer 1000 EUR of my own money to the first person to find a security hole from Dovecot."*

During this project, the Cure53 security investigations aimed at a thorough and in-depth assessment of the selected parts of the Dovecot codebase. The scoped components should be viewed as top priorities in further safe development, and, as such, they pertained mostly to the POP and IMAP protocol stacks, SSL wrapper, implementations of *dcrypt* and *GUID,* as well as other items elaborated on later in this report.

The tests proceeded in two distinct phases, with the first one dedicated to a manual code review, and the second one centered on a penetration test. The latter component took advantage of several running instances of Dovecot. The tests largely sought to spot any vulnerabilities that could allow malicious attackers to benefit from memory corruption, information leaks, authentication bypasses, and logical flaws. In scope was the 2.2.26.0 version of the Dovecot email server software suite, which was released on October 28, 2016. The date coincided with the start date of the Cure53 assignment[2].

Despite much effort and thoroughly all-encompassing approach, the Cure53 testers only managed to assert the excellent security-standing of Dovecot. More specifically, only

[1] http://www.dovecot.org/security.html
[2] http://www.dovecot.org/doc/NEWS

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

three minor security issues have been found in the codebase, thus translating to an exceptionally good outcome for Dovecot, and a true testament to the fact that keeping security promises is at the core of the Dovecot development and operations.

It has to be mentioned that the analyzed code can be only characterized as highly complex. Many parts are commonly interwoven and entangled together in the code base, which makes it particularly hard to grasp. Against this backdrop, one should see the audit process as slightly hindered in terms of the speed and pace it allows to external security investigators. This was especially visible for the API code, while other areas appeared to be less entwined and much more readable for the auditors.

Regardless of complexity, most of the issues and concerns prompted by the code have ultimately turned out non-exploitable. For the most part, they have not warranted creation of dedicated issue-tickets. It is important to underscore that even the areas which looked exploitable at first glance were equipped with an important value of being checked properly, thus having all attack potential successfully mitigated.

It is noticeable that Dovecot has already received a lot of scrutiny regarding its code security. For a complex piece of software that Dovecot constitutes, it is an extremely rare result to stand strong with so few problems. As the findings are few and far between, the report can discuss both test methodologies in considerable detail and elaborate on the logic and sequence of steps within the tests. Only then, it will move on to the coverage of the three spotted weaknesses and their corresponding fix recommendations.

Scope

- **Dovecot Sources & Signatures**
  - http://dovecot.org/releases/2.2/dovecot-2.2.26.0.tar.gz
  - http://dovecot.org/releases/2.2/dovecot-2.2.26.0.tar.gz.sig
  - (released on 28th of October 2016[3])
- **General Info**
  - http://www.dovecot.org/security.html
- **More detailed Scope**
  - The POP and IMAP protocol stacks
  - The process architecture and particularly the login process
  - The User/Password MySQL and LDAP plugins
  - The internal dcrypt encryption API wrapper
  - The GUID implementation

[3] http://www.dovecot.org/list/dovecot-news/2016-October/000329.html

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

Test Methodology

The following section describes the methodology that was used during this source code audit and penetration test. The test was divided into two phases with specific two-fold goals and focal points in scope. The first phase concentrated mostly on manual source code reviews. These reviews aimed at spotting insecure code constructs with a capacity to lead to memory corruption, information leakage or other similar flaws. The second phase of the test was dedicated to classic penetration tests, which examined whether the security promises made by Dovecot in fact hold in the real-life attack situations.

Part 1 (Manual Code Auditing)

A list of items below seeks to detail some of the noteworthy steps undertaken during the first part of the test, which entailed the manual code audit against the sources of the Dovecot software in scope. This is to underline that, in spite of the relatively low number of findings, substantial thoroughness was achieved and considerable efforts have gone into this test. The completed steps are listed next.

- The *doc/documentation.txt* files were studied. It was verifying whether the content of *doc/securecoding.txt* is sane. While this component was found rather complete, it lacked a few points, e.g. regarding the implementation of the *refcounter* assertions[4]
- Existing mitigation against standard memory corruption attacks was analyzed. Using secure wrapper functions instead of direct memory access is considered best practice in this realm. One issue regarding the protection against format string exploits was discovered in this area and is documented in DOV-01-001. Heap management appeared reasonable. The additional hardening against *0-sized* and large allocations inside the *mempool* were certainly considered useful and appropriate. Similarly, making sure that memory is always *zero-initialized* paired with the strategy of having the *free'd* pointers nulled, effectively aided the prevention of additional problems. Another idea pertinent to hardening and aimed at increasing general memory safety was added to the report.
- The checks of the default *Makefiles* revealed them to be lacking all sorts of hardening flags. These should be enabled per default and must be added instead of having distributions apply their compilation option. The alternative approach is described in DOV-01-002.
- At this point, the investigations switched to the actual source code auditing but stayed limited to the agreed scope and the protocols that are defined in it (i.e. POP3/IMAP).
- A top-down approach was first utilized in order to perform the so called tracing of the flow in terms of user-controlled data. Further, checking if it went through

[4] https://en.wikipedia.org/wiki/Reference_counting

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

vulnerable sinks was also performed. In this realm, focus was mostly on the classic memory corruption issues and especially encompassed verifications of that code parts that are marked as *UNSAFE* and reassurance that these do not pose a security threat. Generally *integer* overflows and bypassing secure API functions with direct memory access, which seemed like a weak point. However, no actual security issue was found here.

- The bottom-up method was envisioned and executed next to find suspicious functions and then trace them back to user-input.
- The overall complexity of the code made auditing increasingly difficult. Considerable amount of time was spent on trying to understand most of the implementation. A reasonable ratio between the invested time and understanding code, as well as allocation to resources to specifically looking for bugs needed to be found. Thus, a number of tasks revolved around on trying to catch the so called low-hanging fruits, which ultimately was not worthwhile.
- Given the simplicity of the POP3 protocol, it was found to be secure in terms of possible memory corruption attacks. In the same vein, there were no logical issues found.
- Analogical findings address the IMAP, although it boasts a lot more complex implementation. Sill, being a text-based protocol it had been examined and determined well-thought in terms of security for Dovecot.
- Security architecture documentation was then reviewed. No obvious problems were observed.
- Having reached this point, the audit moved to a closer investigation of the source code with respect to the cryptographic components of the implementation. The scope was still limited here to the defined aspects.
- Looking at the involved base-libraries ensued, the usually problematic buffers, arrays and strings were subject to logical abstracting.
- Relatively stringent usage of assertions was verified, thus making the debugging builds less vulnerable and consequently more verifiable.
- SSL wrapper implementation was partially audited; the tests involved primarily its relations with the supported libraries.
- Command line tools were analyzed with the purpose of acquiring a better understanding of the general process management and administrative instrumentation.
- Source code was checked for *integer* truncation and *signedness* issues specific to 64-bit systems. This item followed the arguments specified in the paper entitled "*Twice the Bits, Twice the Trouble*"[5]. It was found that *size_t* was used in all cases relevant to this issue.
- The implementation of the cryptographic library *dcrypt* was audited, yet found in good standing, straightforward and solid.

[5] https://www.sec.cs.tu-bs.de/pubs/2016-ccs.pdf

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

Part 2 (Code-Assisted Penetration Testing)

A list of items below seeks to detail some of the noteworthy steps undertaken during the second part of the test, which encompassed code-assisted penetration testing against the Dovecot software in scope. Given that the manual source code audit did not yield an overly large amount of findings, the second approach was chosen as an additional measurement for maximizing the test coverage. As for specific steps executed to enrich this phase, these can be found listed and discussed in the following bullet points.

- The manual audit was not particularly fruitful, especially in terms of lacking extensive coverage. For this reason, a code-assisted method was chosen to broaden the approach and increase the validity of the assessment's results.
- As a result, a Dovecot POP3 and IMAP servers were set up and all tests were based on the running processes.
- Mainly short *printf* debugging and process inspections were employed as means to quickly find interesting code parts and user-controlled data.
- Using this method, the login process was covered in detail, including possible manipulation of the *userdb* and *passdb*. The tests have additionally relied on fuzzing.
- Since no further issues were found, the attack surface was expanded to also include world-writable *unix* domain sockets which are used by the plugins to communicate with the master process. The reasoning behind that was that local attackers could benefit from exploiting them.
- The code that handles the aforementioned communication over *unix* domain/IPC sockets and relevant for the master process was found less abstract and likely more vulnerable. However, upon further manual auditing and testing, it was eventually deemed to have the same level of robustness as the main protocols.
- Test harness was built to achieve more control over code execution and possibly untested exceptional behavior.
- A decision was made to focus on certificate-based authentication.
- In subsequent steps, a certificate authority infrastructure was created as a basis for further investigation of the SSL/TLS authentication implementation.
- Permuted client-side certificates trying to force unexpected behavior were used. Nevertheless, no irregular patterns were observed.
- Client-side implementations like *s_client* and *claws-mail* were instrumented to demonstrate how authenticating via certificates occurs. This served as a basis for more detailed investigations which nonetheless yielded no irregularities.
- Authentication via external services (MySQL/LDAP) was checked with primary focus placed on memory corruptions. Once again no bugs were found in this realm as well.

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

Identified Weaknesses

The following sections list both weaknesses 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 weakness is additionally given a unique identifier (e.g. *DOV-01-001*) for the purpose of facilitating any future follow-up correspondence.

DOV-01-001 Format String Protection can be bypassed (*Low*)

Dovecot always tries to follow the principle of multiple layers of security. It does not simply rely on input validation to make sure that vulnerabilities do not arise easily but rather attempts to eradicate certain bug classes at the very core of some API functions. One of these mechanisms is the detection of possible format string exploits that abuse the "*%n*" format parameter in hopes of arbitrarily writing to memory. The function that is used internally to detect such cases is depicted in the following excerpt.

File:
/dovecot-2.2.26.0/src/lib/printf-format-fix.c

Affected Code:

```
static const char *
printf_format_fix_noalloc(const char *format, unsigned int *len_r)
{
	const char *p;
	const char *ret = format;

	for (p = format; *p != '\0'; ) {
		if (*p++ == '%') {
			switch (*p) {
			case 'n':
				i_panic("%%n modifier used");
			case 'm':
				if (ret != format)
					i_panic("%%m used twice");
```

A problem with this approach has been discovered. In essence, while the function in question can successfully detect the usage of "*%n*", it is easily bypassable. This can be done by simply using "*%1$n" or "%hn*" as format parameter, because the check only verifies whether the "*n*" character directly follows the "*%*" character. As a result, both values have the same meaning: writing the number of bytes that were printed to the first specified pointer. *Glibc,* however, is also able to detect the misuse of direct parameter access when *FORTIFY_SOURCE* is employed and tends to bail out accordingly. At the

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

end, because this protection has also been bypassed in the past[6], it is recommended to implement some additional hardening to *printf_format_fix_noalloc* and detect the usage of "*%...n*" rather than "*%n*" only.

DOV-01-002 Default Makefile fails to add Hardening Flags (*Low*)

Using tools like *checksec*[7] or *PEDA*'s[8] built-in functionality to check for basic hardening support reveals that the default compiler options omit *PIE*[9], full *RELRO*[10] and stack canaries when building Dovecot from a source:

```
dovecot-2.2.26.0$ gdb src/master/.libs/dovecot
Reading symbols from src/master/.libs/dovecot...done.
(gdb) checksec
CANARY    : disabled
FORTIFY   : disabled
NX        : ENABLED
PIE       : disabled
RELRO     : disabled
```

On the one hand, *PIE* renders the exploitation of memory corruption vulnerabilities a lot more difficult. This can be attributed to the fact that additional information leaks are required to conduct a successful attack. *RELRO,* on the other hand, masks different binary sections, like the *GOT,* as read-only. Therefore, it kills a handful of techniques that come in handy when attackers are able to arbitrarily overwrite memory. The tests showed that enabling these features had almost no impact, neither on the performance, nor on the general functionality of Dovecot. This is why it is recommended to add the necessary compiler flags to the generated Makefile:

```
dovecot-2.2.26.0$ make CFLAGS='-Wl,-z,relro,-z,now -pie -fPIE -fstack-protector-
all -D_FORTIFY_SOURCE=2 -O1'
[...]
dovecot-2.2.26.0$ gdb src/master/.libs/dovecot
Reading symbols from src/master/.libs/dovecot...(no debugging symbols
found)...done.
(gdb) checksec
CANARY    : ENABLED
FORTIFY   : ENABLED
NX        : ENABLED
PIE       : ENABLED
RELRO     : FULL
```

[6] http://phrack.org/issues/67/9.html
[7] http://www.trapkit.de/tools/checksec.html
[8] https://github.com/longld/peda
[9] https://gcc.gnu.org/onlinedocs/gcc/Code-Gen-Options.html
[10] http://tk-blog.blogspot.de/2009/02/relro-not-so-well-known-memory.html

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

DOV-01-003 Memorypool Allocator fails to check for *Integer* Overflows (*Low*)

Another addition to the “defense in-depth” philosophy already embraced by the Dovecot suite concerns the wrapper functions inside the memory pool allocator. The idea to wrap all standard memory allocation functions that are offered by *Glibc* into wrappers, which then extend the functionality and security, should be seen as quite positive and plausible. For example, Dovecot implements additional checks for *malloc* or *calloc* in *pool_system_malloc* by explicitly disallowing *zero-sized* or overly large allocations. Additionally, it always defaults to *calloc,* making sure that memory is always zero-initialized, unless *Boehm GC* is used.

The slight issue is that this type of wrapping quite often leads to the shape of code for which the programmers must check for *integer* overflows. This holds especially when s/he is trying to allocate memory for an array of items characterized by each component having the same size. Instead of being required to write code like *malloc(size * num_elements),* it makes more sense to have a wrapper function that allows to write *size_malloc(size, num_elements)*. The former code structure is prone to an overflow, which can happen before *pool_system_malloc* is entered and may result in allocations that are too small. In this case, the multiplication of *size* and *num_elements* should be checked internally, thus making sure that an overflow of the type described above can in fact never happen.

GCC provides a good list of built-ins for safe arithmetics around *integrer.* These include *__builtin_mul_overflow_p* or *__builtin_add_overflow*[11]. Another strategy would be to implement something similar to PHP’s *safe_emalloc*[12] where the arguments are verified with architecture-dependent code in *zend_safe_address*[13].

It is recommended to implement and make use of the specified alternative wrapper function since doing so would slightly increase the general security of Dovecot.

[11] https://gcc.gnu.org/onlinedocs/gcc/Integer-Overflow-Builtins.html
[12] http://php.net/manual/de/internals2.memory.management.php
[13] https://github.com/php/php-src/blob/master/Zend/zend_multiply.h

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

Conclusion

The overall very much positive outcome of this security assignment performed by four testers from the Cure53 team can be inferred from the minimal number of discoveries in the context of the application's high-complexity, as well as a very extensive and in-depth coverage. As for the latter, a considerable length of twenty days of testing over the two months of October and November of 2016 attest to a near-impenetrable security disposition of the Dovecot suite.

Quite clearly, this is a refreshingly pleasant result, which should by no means be taken-for-granted, or perceived as the "usual standard" in the mature and complex software environments of similar kind. At the same time, it has to be noted though that the general Dovecot code base is massive, so the scope was limited to the most commonly used and deployed components. In addition, the complexity in certain parts of the code base initially made it very hard to uncover and understand the logic of all entanglements. The level of complexity was not even across the code base, but rather affected the API are most profusely. Conversely, other parts posed no such difficulty to the auditors. Besides these minor struggles at the early stage, the audit managed to achieve proper coverage of the given scope.

Finally, as with all software, excellent results do not mean that there is nothing left to do. In fact, it is a clear and vocal recommendation of the Cure53 testers' part to engage in security testing against the components of Dovecot that were not in the primary scope of this test. This strategy of incorporating more areas through expansion could help ensure that the positive impression translates into other areas and persists, even when one imagines the possible effects of the less common usage scenarios.

In sum, there is no doubt that the Dovecot email server software holds strong and robust, even when faced with a very stern and in-depth look into its codebase.

Cure53 would like to thank Gervase Markham and Chris Riley of Mozilla for their excellent project coordination, support and assistance, both before and during this assignment. Cure53 would further like to extend gratitude to Neil Cook & Timo Sirainen, two maintainers of the Dovecot project, for their help during the scoping phase of this assessment.