Global Information Assurance Certification Paper

The Role of Static Analysis in Hardening Open Source Intrusion Detection Systems

GIAC (GCIA) Gold Certification

Author: Jeff Sass, jsass@adobe.com
Advisor: Stephen Northcutt
Accepted: March 28, 2016

Abstract

When deploying an open source Intrusion Detection System (IDS) into a network, it is critical to harden it against attackers. An IDS is designed to detect attacks instead of inadvertently enabling them. One approach to assist in this effort is to use static code analysis on the source code of the IDS. This paper details how to use Coverity's static analysis tools on the Security Onion distribution to find security vulnerabilities. A look at Coverity's security code checkers, with a focus toward UNINIT, BUFFER_SIZE, and USE_AFTER_FREE is discussed.

1. Introduction

Intrusion analysts use the principles of network security monitoring (NSM) to help secure computer systems. NSM is "the collection, analysis and escalation of indications and warnings to detect and respond to intrusions" (Bejtlich, 2013). NSM core functions include intrusion detection systems (IDS), network based IDS (NIDS), host intrusion detection systems (HIDS), and physical intrusion detection systems (Physical IDS) (Berge, n.d). Analysts should evaluate software packages such as IDS and HIDS before deploying them.

There are many different ways to determine how secure a given software package is. One way is to use Aberlarde's security systems engineering approach (Abelarde, 2016). This approach details how commercial and open source software packages are evaluated at each phase of the software development life cycle (SDLC) to determine their security profiles. An advantage of examining open source software is direct access to the code. With direct access, developers can use techniques such as code inspection and static code analysis.

Static code analysis (SCA) is a way of finding issues in software without executing it. The SCA tool accomplishes this by emulating the execution of the different branches of the code by using possible input data. The SCA tool reveals both quality issues (e.g. COPY_PASTE_ERROR, FORWARD_NULL, INCOMPATIBLE_CAST) and security issues (e.g. UNINIT, BUFFER_SIZE, and USE_AFTER_FREE). The SCA tool also shows specific fixes the developer can apply to the source code to reduce the software's defect density. It calculates defect density by dividing the number of defects by the size of the component (usually specified in lines of code). In 2014, the average defect density for open source software was 0.61 per thousand lines of code or KLOC. In contrast, commercial software's defect density was 0.76 per KLOC (Coverity, 2014).

There are many static analysis tools to choose from (OWASP, 2016). Coverity has been gaining popularity in the last ten years after the Coverity Scan service was made available (Coverity, 2016). Coverity Scan allows open source developers a way to submit their code to Coverity's cloud-based service for analysis and examine the results free of charge. Coverity also offers the same analysis tools in a commercial product that can be

deployed locally in their customer's environment. This paper demonstrates how Coverity's static code analysis can be used in both deployment scenarios to scan some of the software packages that make up the Security Onion distribution.

2. Security Onion

Security Onion is a Linux distribution maintained by Doug Burks that includes full packet capture, NIDS, HIDS, and a set of analysis tools (Burks, n.d.). Those tools include:

- netsniff-ng for full-packet capture
- Snort, Suricata and Bro for NIDS
- OSSEC for HIDS
- Sguil, Squert, Snorby, and ELSA, for data analysis

Using the Security Onion distribution saves time when compared with configuring each of the tools separately. Before starting development with this distribution, step-by-step instructions for installing, configuring, and updating Security Onion should be followed (Burks, 2016). Once that is complete, the developer can examine the source code of the Security Onion software packages to look for security vulnerabilities.

3. Coverity Scan

The first deployment option for Coverity is Coverity Scan. Coverity Scan is a cloud service where registered open source developers upload their source code for analysis. The Coverity static analysis engine then executes against that source code. Developers review the reported issues, follow the advice to fix the issues, and then re-submit the source code. Coverity Scan is free to the open source community.

3.1. Coverity Scan Overview

Coverity Scan started in 2006 as a project funded by the Department of Homeland Security. The main mission was to improve the quality of open source software that the

nation was beginning to use. The funding lasted three years until 2009 when Coverity took full ownership of the project (J. Croall, personal interview, January 21, 2016).

The Linux operating system was one of the original Coverity Scan users. Currently, there just under 7,000 projects with over 15,000 individual users using Coverity Scan. Some of the projects include Python, OpenSSL, PHP as well as packages found in Security Onion like Snort, Bro, and Wireshark. To prevent overloading the servers, Coverity Scan limits the number of uploads on projects with large codebases. Developers are permitted to submit up to three builds a day and twelve builds per week if their software package has less than 100,000 lines of code (Frequently Asked Questions, 2016). To help increase the security profile of one of the projects, developers request contributor access from the maintainers.

3.2. Coverity Scan Example: Wireshark

Developers follow a four-step process when using Coverity Scan: build; analyze; commit defects; and review results. For the build step, pass the native build command as an argument to Coverity's command line cov-build tool. Cov-build instruments the native build and stores the information in the intermediate directory specified with the --dir flag. Using Wireshark as an example, the Coverity compile command would be:

```
$ cov-build --encoding UTF-8 \
  --dir ~/cov-inter-wireshark make
```

For the analysis step, upload the intermediate directory to Coverity Scan manually or with a continuous integration system (i.e. Travis-CI). Code analysis is performed on the Coverity servers as opposed to locally on the developer's system. For the commit defects step, Coverity handles this automatically. To review the results, log on to the Coverity Connect web interface where the defects are shown inline with the source code. Section 5 below details this process.

The Wireshark project is an active user of Coverity Scan. They have fixed thousands of defects since 2006 and have a very low defect density of 0.26 per KLOC as

shown in Figure 1.

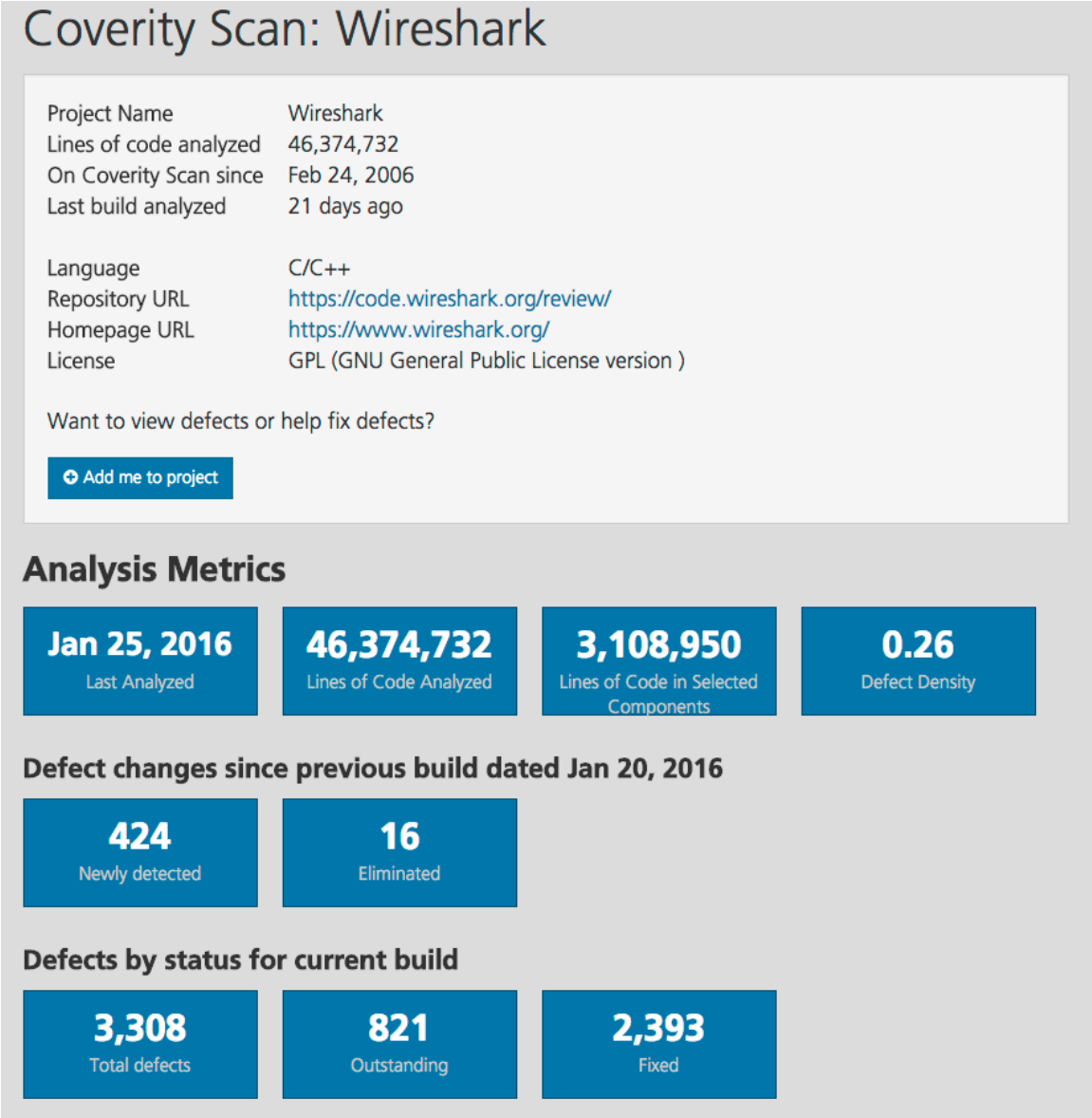

Figure 1: Coverity Scan: Wireshark

(https://scan.coverity.com/projects/wireshark)

4. Coverity Local Analysis

In contrast to Coverity Scan's cloud service, developers can choose to purchase Coverity's commercial offering. The commercial offering runs locally on their network. A standard Coverity deployment uses two machines in a client/server architecture.

Security Onion is the local development machine and acts as the client which sends the results to the Coverity database server. By default, Security Onion's software packages are installed as executables. Developers must compile and analyze the corresponding source code by downloading it first. Appendix A lists each of the commands to install Coverity, the GCC compiler, and the source code of the software under investigation. Developers execute the code analysis on the client machine rather than using Coverity Scan's servers. The database that stores the results is on a local network instead of on a Coverity Scan server. Browsing the results is done by logging into the Coverity web server and selecting the appropriate project (i.e. Wireshark) as shown in Figure 2.

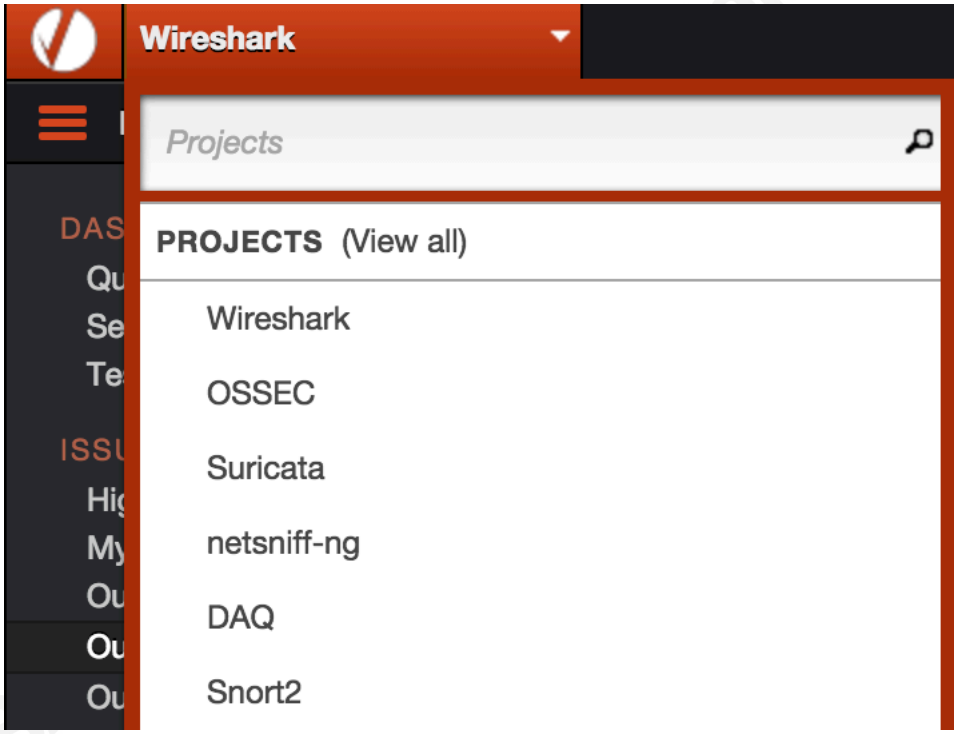

Figure 2: Coverity Project Menu

Once a project is selected, select the Coverity Menu (the three-line icon) and choose “Outstanding Security Risks” as shown in Figure 3.

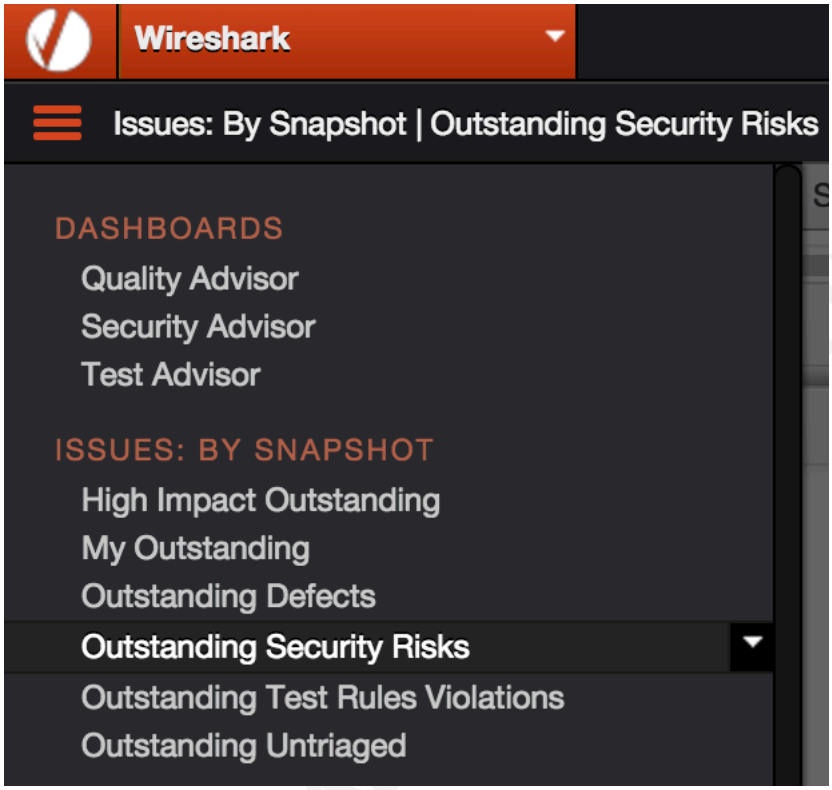

Figure 3: Outstanding Security Risks Filter

This view filters all of the Coverity defects into a smaller list that only includes the security issues. The examples in the next section use this filter.

5. Fixing Security Vulnerabilities

Before fixing the code, let's examine how the "do no harm rule" can be applied to software as well as how compiler warnings fit into the static analysis picture.

5.1. Do No Harm

"Learning to write clean code is hard work" (Martin 2009). In the beginning, source code can be elegant, but as time passes it can become "increasingly sucky" (Skorkin, 2010). Reading source code that one did not write is a critical part of being a good developer. For intrusion analysts who might not be as familiar with reading and writing code, it can be a daunting task.

> *"The Boy Scouts of America have a simple rule that we can apply to our profession. Leave the campground cleaner than you found it. If we all checked-in our code a little cleaner than when we checked it out, the code simply could not rot" (Martin, 2009).*

There are two benefits to the "do no harm rule": developers improve their coding skills; and the original authors will appreciate the responsible disclosure (Hughes, 2015).

5.2. Compiler Warnings

Another aspect of static code analysis is compiler warnings. Getting code to compile is a mini-celebration in itself, so compiler warnings are often ignored. Taking an example from the daq-2.0.6 package, line 859 of daq_afpacket.c declares the variable `rc`:

```
int rc;
```

Line 866 contains:

```
rc = send(instance->peer->fd, NULL, 0, 0);
```

The compiler warning is:

```
daq_afpacket.c:859:25: warning: variable 'rc' set but not used
[-Wunused-but-set-variable]
                    int rc;
                        ^
```

The compiler is informing the developer that the return value from the call to `send()` is set in the variable `rc`, but `rc` is not used later in the function. One fix would be to delete line 859 and change line 866 to:

```
(void) send(instance->peer->fd, NULL, 0, 0);
```

This change silences the compiler warning and keeps the code change as close to the original as possible. By assigning the return value from the `send()` call to `(void)`, the code is ignoring it as it does currently. Another possible fix is to add additional code after line 866, to check `rc` against all of the return values. That fix changes the program execution and should be reviewed by the maintainers.

Compilers also have the ability to "treat warnings as errors". Turning on this feature, is a good way to introduce a level of coding discipline in a phased approach to the project. Developers can turn on one warning at a time, fix each one, and then turn on additional warnings when time permits. On Adobe Photoshop, the compilers have the option "treat warnings as errors" turned on which forces a higher level of awareness amongst the team. The continuous build system fails the build with newly introduced compiler warnings. If the build fails, the team fixes the errors quickly. Another reason to turn on "treat warnings as errors", is to minimize static analysis defects. It is better to eliminate them from the code with the compiler's help before adding another tool.

5.3. Coverity Security Checkers

Coverity 7.7 has over seventy checkers that apply to C and C++ and of these, eighteen focus on security issues. This section focuses on UNINIT, BUFFER_SIZE, and USE_AFTER_FREE.

5.3.1. UNINIT

In ANSI C, the "initial contents of a variable are undefined" (Roberts, 1997). Because the language allows the definition of variables without initialization, there is often a large amount of C code that doesn't explicitly initialize variables. Some of that code immediately fill the variable after its declaration, so initialization does happen. Sometimes the compiler sets it to zero automatically. Because developers have to remember these rules, there is room for security issues to enter the software. Although there have been proposals to fix this for the C language, for now, developers need to remember the rules (Myers, 2015).

One way to eliminate these issues is to use Coverity's security checker UNINIT. UNINIT looks for uninitialized stack variables and dynamically allocated memory on the heap that could lead to crashes or security issues. Line 222 of sf_bpf_filter.c in the daq-

2.0.6 package declares an array of int32's called `mem`.

```
214 DAQ_SO_PUBLIC u_int bpf_filter(pc, p, wirelen, buflen)
215     register const struct bpf_insn *pc;
216     register const u_char *p;
217     u_int wirelen;
218     register u_int buflen;
219 {
220     register u_int32 A, X;
221     register int k;
        1. var_decl: Declaring variable mem without initializer.
222     int32 mem[BPF_MEMWORDS];
```

Figure 4: `mem` declaration

Line 406 uses `mem` in a condition where it was not initialized.

```
        125. Continuing loop
399                 continue;
400
401             case BPF_LDX | BPF_IMM:
402                 X = pc->k;
403                 continue;
404
405             case BPF_LD | BPF_MEM:
        CID 15144 (#2 of 2): Uninitialized scalar variable (UNINIT)
        128. uninit_use: Using uninitialized value mem[pc->k] .
406                 A = mem[pc->k];
407                 continue;
```

Figure 5: `mem` assignment

Coverity simulated running through the loop exercising all code paths, as shown in the green text. The simulation found that in at least one condition, the variable `mem` was assigned to variable `A` before initialization. To fix this issue, explicitly zero-initialize the array in line 222 as follows.

```
int32 mem[BPF_MEMWORDS] = {0};
```

5.3.2. BUFFER_SIZE

According to Michael Howard and David LeBlanc's *Writing Secure Code*, "buffer overruns that lead to a security patch can cost up to $100,000" (Howard, 2003). Coverity's security checker BUFFER_SIZE helps developers find and fix defects that involve buffers in their C/C++ code. Taking an example from the snort-2.9.8.0 package, line 962 of encode.c initializes the variable `next` of type `PROTO_ID` to `PROTO_MAX`. `PROTO_MAX` is the last element of the `PROTO_ID` enum defined as:

```
typedef enum {
  PROTO_TCP
  PROTO_UDP
  ...
  PROTO_MAX
} PROTO_ID;
```

Line 960 defines the function `UDP_Encode` as shown in Figure 6.

```
960 static ENC_STATUS UDP_Encode (EncState* enc, Buffer* in, Buffer* out)
961 {
962     PROTO_ID next = PROTO_MAX;
963
        1. Condition enc->layer < enc->p->next_layer, taking true branch
964     if ( enc->layer < enc->p->next_layer )
965     {
966         next = enc->p->layers[enc->layer].proto;
967     }
        2. Condition enc->type == ENC_UDP, taking false branch
        3. Condition PROTO_GTP == next, taking true branch
        4. Condition encoders[next].fencode, taking true branch
968     if ( enc->type == ENC_UDP || ((PROTO_GTP == next) && (encoders[next].fencode)) )
969     {
970         int len;
971         ENC_STATUS err;
972         uint32_t start = out->end;
973
974         UDPHdr* hi = (UDPHdr*)enc->p->layers[enc->layer-1].start;
975         UDPHdr* ho = (UDPHdr*)(out->base + out->end);
        5. Condition out->end > out->size, taking false branch
976         UPDATE_BOUND(out, sizeof(*ho));
977
        6. Condition enc->flags & 0x80000000U, taking true branch
978         if ( FORWARD(enc) )
979         {
980             ho->uh_sport = hi->uh_sport;
981             ho->uh_dport = hi->uh_dport;
        7. Falling through to end of if statement
982         }
983         else
984         {
985             ho->uh_sport = hi->uh_dport;
986             ho->uh_dport = hi->uh_sport;
987         }
988
        8. Condition enc->type != ENC_UDP, taking true branch
989         if ( enc->type != ENC_UDP)
990         {
        9. return_constant: Function call NextEncoder(enc) returns 22.
        10. assignment: Assigning: next = NextEncoder(enc). The value of next is now 22.
991             next = NextEncoder(enc);
        CID 15213 (#1 of 1): Out-of-bounds read (OVERRUN)
        11. overrun-local: Overrunning array encoders of 22 24-byte elements at element index 22 (byte offset 528) using index next (which evaluates to 22).
992             err = encoders[next].fencode(enc, in, out);
993             if (ENC_OK != err ) return err;
994             len = out->end - start;
995         }
```

Figure 6: out-of-bounds read example

The green text shows which execution path Coverity used. The value returned from the `NextEncoder` function is stored in `next` which is of type `PROTO_ID`. There is a case where the returned value could be `PROTO_MAX`, or 22, which is the last element of the `enum`. Line 992 indexes into the encoders array at the position specified in `next` which is one past the end of the array because array indexing starts at 0 instead of 1. To prevent this possible buffer overrun, wrap line 992 in an if/else statement to check that `next` is less than `PROTO_MAX` before it is used to index into the `encoders` array:

Jeff Sass, jsass@adobe.com

```
if (next < PROTO_MAX)
{
    err = encoders[next].fencode(enc, in, out);
}
else
{
    err = ENC_BAD_PROTO;
}
```

5.3.3. USE_AFTER_FREE

Defining variables reserves a place in memory for them. When the program explicitly frees the memory, developers need to ensure there are no cases where the memory is used after it is freed. Using memory in this way can lead to unpredictable results and possible exploitation.

One way to eliminate these issues is to use Coverity's security checker USE_AFTER_FREE. Taking an example from the netsniff-ng-0.6.0 package, line 304 of curvetun_client.c declares a pointer to a structure called `ahead` as shown in Figure 7.

```
300 int client_main(char *home, char *dev, char *host, char *port, int udp)
301 {
302         int fd = -1, tunfd = 0, retry_server = 0;
303         int ret, try = 1, i;
304         struct addrinfo hints, *ahead, *ai;
305         struct pollfd fds[2];
306         struct curve25519_proto *p;
307         struct curve25519_struct *c;
308         char *buff;
309         size_t blen = TUNBUFF_SIZ; //FIXME
```

Figure 7: netsniff-ng - `ahead` declaration

Line 339 assigns the `ahead` pointer to `ai` as shown in Figure 8.

```
338
    12. alias: Assigning: ai = ahead. Now both point to the same storage.
    13. Condition ai != NULL, taking true branch
    14. Condition fd < 0, taking true branch
    17. Condition ai != NULL, taking true branch
    18. Condition fd < 0, taking true branch
    22. Condition ai != NULL, taking false branch
    CID 21017 (#1 of 1): Use after free (USE_AFTER_FREE)
    31. use_after_free: Using freed pointer ahead.
339         for (ai = ahead; ai != NULL && fd < 0; ai = ai->ai_next) {
340                 fd = socket(ai->ai_family, ai->ai_socktype, ai->ai_protocol);
    15. Condition fd < 0, taking true branch
    19. Condition fd < 0, taking false branch
341                 if (fd < 0)
    16. Continuing loop
342                         continue;
343                 ret = connect(fd, ai->ai_addr, ai->ai_addrlen);
    20. Condition ret < 0, taking true branch
344                 if (ret < 0) {
345                         syslog(LOG_ERR, "Cannot connect to remote, try %d: %s!\n",
346                                 try++, strerror(errno));
347                         close(fd);
348                         fd = -1;
    21. Continuing loop
349                         continue;
350                 }
351
352                 set_socket_keepalive(fd);
353                 set_mtu_disc_dont(fd);
354                 if (!udp)
355                         set_tcp_nodelay(fd);
356         }
357
    23. freed_arg: freeaddrinfo frees ahead.
358         freeaddrinfo(ahead);
359
    24. Condition fd < 0, taking true branch
360         if (fd < 0) {
361                 syslog(LOG_ERR, "Cannot create socket! Retry!\n");
362                 curve25519_tfm_free(c);
363                 fd = -1;
364                 retry_server = 1;
365                 closed_by_server = 0;
366                 sleep(1);
    25. Jumping to label retry
367                 goto retry;
368         }
```

Figure 8: `ahead` pointer assignment

Coverity found in line 358 the `ahead` pointer was freed. The `goto` statement at line 367 jumps program execution back to line 311. The next time through the loop at line 339, the pointer was assigned to `ai` without first checking the pointer for `NULL`. To fix the issue, add the following line of code after line 358 to set the pointer to `NULL`.

```
ahead = NULL;
```

Jeff Sass, jsass@adobe.com

6. Responsible Disclosure

After fixing vulnerabilities, developers have an ethical responsibility to disclose the issues back to the maintainers of the code. For projects like Wireshark that use GitHub, fixes are submitted with a "git push" command by following the project documentation (Wireshark Developer's Guide, 2014). Other projects have mailing lists or defect tracking systems to submit the fixes.

7. Future Work

In January 2016, Coverity released version 8.0 of its static analysis tools. One of the major new features was the ability to analyze Python code. Security Onion contains a packet manipulation tool called Scapy. Scapy is gaining in popularity especially as intrusion analysts investigate devices that make up the Internet of Things (The 2015 SANS Holiday Hack Challenge, 2015). A future project could examine the static code analysis results of Scapy.

8. Conclusion

Hardening computer networks with an open source IDS requires the intrusion analyst understand the security profile of the software packages on the system. By utilizing static code analysis on the software that makes up the IDS, the analyst has a better understanding of the security profile the open source software provides. John Carmack, the co-founder of id Software, stated:

> *"The most important thing I have done as a programmer in recent years is to aggressively pursue static code analysis" (Carmack, 2011).*

Taking the advice from one the most famous software developers allows the intrusion analyst to utilize some of the best practices from software developers.

Jeff Sass, jsass@adobe.com

References

Abelarde, J. (2016, January 25). Security Systems Engineering Approach in Evaluating Commercial and Open Source Software Products. Retrieved February 2, 2016, from https://www.sans.org/reading-room/whitepapers/OpenSource/security-systems-engineering-approach-evaluating-commercial-open-source-software-products-36687

About Coverity Scan. (n.d.). Retrieved February 08, 2016, from https://scan.coverity.com/about

Bejtlich, R. (2013). The practice of network security monitoring: Understanding incident detection and response. San Francisco: No Starch Press.

Berge, M. (n.d.). Intrusion Detection FAQ: What is Intrusion Detection? Retrieved February 19, 2016, from https://www.sans.org/security-resources/idfaq/what_is_id.php

Burks, D. (n.d.). Security Onion. Retrieved February 2, 2016, from https://github.com/Security-Onion-Solutions/security-onion

Burks, D. (2016, January 20). Security Onion Installation. Retrieved February 09, 2016, from https://github.com/Security-Onion-Solutions/security-onion/wiki/Installation

Carmack, J. (2011, December 27). In-Depth: Static Code Analysis. Retrieved February 18, 2016, from http://www.gamasutra.com/view/news/128836/InDepth_Static_Code_Analysis.php

Coverity Scan Open Source Report Shows Commercial Code Is More Compliant to Security Standards than Open Source Code - Coverity. (2015, July 29). Retrieved February 05, 2016, from http://www.coverity.com/press-releases/coverity-scan-open-source-report-shows-commercial-code-is-more-compliant-to-security-standards-than-open-source-code/

Coverity Scan: Wireshark. (n.d.). Retrieved February 15, 2016, from https://scan.coverity.com/projects/wireshark

Frequently Asked Questions (FAQ). (n.d.). Retrieved February 15, 2016, from https://scan.coverity.com/faq/

Jeff Sass, jsass@adobe.com

Howard, M., & LeBlanc, D. (2003). Public Enemy #1: The Buffer Overrun. In *Writing Secure Code*. Redmond, WA: Microsoft Press.

Hughes, M. (2015, September 15). Full or Responsible Disclosure: How Security Vulnerabilities Are Disclosed. Retrieved February 15, 2016, from http://www.makeuseof.com/tag/responsible-disclosure-security-vulnerabilities/

OWASP. (2016, January 16). Source Code Analysis Tools. Retrieved February 6, 2016, from https://www.owasp.org/index.php/Source_Code_Analysis_Tools

Martin, R. C. (2009). [Introduction]. *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River, NJ: Prentice Hall.

Martin, R. C. (2009). Clean Code. In *Clean Code: A Handbook of Agile Software Craftsmanship*. (pp. 1-15) Upper Saddle River, NJ: Prentice Hall.

Myers, S. (2015, November 13). Breaking all the Eggs in C. Retrieved February 18, 2016, from http://scottmeyers.blogspot.com/2015/11/breaking-all-eggs-in-c.html

Roberts, E. (1997). An Overview of ANSI C. In *Programming abstractions in C: A Second Course in Computer Science*. Reading, MA: Addison Wesley.

Skorkin, A. (2010, May 19). Retrieved February 08, 2016, from http://www.skorks.com/2010/05/why-i-love-reading-other-peoples-code-and-you-should-too/

The 2015 SANS Holiday Hack Challenge. (2015, December). Retrieved February 18, 2016, from https://www.holidayhackchallenge.com/

Wireshark Developer's Guide. (2014, December 28). Retrieved February 18, 2016, from https://www.wireshark.org/docs/wsdg_html_chunked/ChSrcContribute.html#ChSrcSend

What is an Intrusion Detection System (IDS)? - Definition from Techopedia. (n.d.). Retrieved February 04, 2016, from https://www.techopedia.com/definition/3988/intrusion-detection-system-ids

Jeff Sass, jsass@adobe.com

Appendix A
Integrating Coverity 7.7 with Security Onion 14.04.3.1

Prerequisites: Install Security Onion 14.04.3.1 per the steps at:
https://github.com/Security-Onion-Solutions/security-onion/wiki/QuickISOImage
Ensure you have a minimum of 8GB of RAM and a 40GB hard drive

```
/* Install development tools, install updates and reboot */
$ sudo apt-get install build-essential
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo reboot

/* Download source code in .tar.gz format with a web browser to
      ~/Downloads with their default names*/
bro-2.4.1.tar.gz
daq-2.0.6.tar.gz
netsniff-ng-0.6.0.tar.gz
ossec-hids-2.8.2.tar.gz
snort-2.9.8.0.tar.gz
suricata-3.0.tar.gz
wireshark-2.0.1.tar.bz2

/* Download Coverity and the license file with a web browser to
      ~/Downloads with their default names*/
cov-analysis-linux64-7.7.0.tar.gz
license.bat

/* Download and Install Coverity at root of user's home directory */
$ cd ~
$ mv ~/Downloads/cov-analysis-linux64-7.7.0.tar.gz .
$ tar xvfz cov-analysis-linux64-7.7.0.tar.gz

/* Install license file to Coverity bin directory */
$ cp ~/Downloads/license.dat ~/cov-analysis-linux64-7.7.0/bin
```

Jeff Sass, jsass@adobe.com

```
/* edit user's .bashrc to update path to include Coverity binaries
  replace "username" with your username */
$ export PATH="$PATH:/home/username/cov-analysis-linux64-7.7.0/bin"
$ . ./.bashrc

/* BRO ---------------------------------------------------------------*/

/* Install prerequisites for Bro */
$ sudo apt-get install python-dev swig

/* Configure and compile Bro with Coverity */
$ cd ~/src
$ cp ~/Downloads/bro-2.4.1.tar.gz .
$ tar xvfz bro-2.4.1.tar.gz
$ cd bro-2.4.1/
$ ./configure
$ cov-build --encoding UTF-8 --dir ~/cov-inter-bro make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze Bro with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-bro --all --enable-callgraph-metrics

/* Commit defects to Coverity stream "bro-mainline"
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream bro-mainline --dir ~/cov-inter-bro --user username

/* DAQ ---------------------------------------------------------------*/

/* Configure and compile DAQ with Coverity */
$ cp ~/Downloads/daq-2.0.6.tar.gz .
$ tar -xvzf daq-2.0.6.tar.gz
$ cd ~/src/daq-2.0.6/
$ cov-build --encoding UTF-8 --dir ~/cov-inter-daq/ make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze DAQ with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-daq/ --all --enable-callgraph-metrics
```

Jeff Sass, jsass@adobe.com

```
/* Commit defects to Coverity stream "daq-mainline"
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream daq-mainline --dir ~/cov-inter-daq/ --user username

/* netsniff-ng ----------------------------------------------------*/

/* Install prerequisites for netsniff-ng */
$ sudo apt-get install ccache flex bison libnl-3-dev \
  libnl-genl-3-dev libnl-route-3-dev libgeoip-dev \
  libnetfilter-conntrack-dev libncurses5-dev liburcu-dev \
  libnacl-dev libpcap-dev zlib1g-dev libcli-dev libnet1-dev

/* Configure and compile netsniff-ng with Coverity */
$ cd ~/src
$ cp ~/Downloads/netsniff-ng-0.6.0.tar.gz .
$ tar xvfz netsniff-ng-0.6.0.tar.gz
$ cd netsniff-ng-0.6.0/
$ ./configure
$ cov-build --encoding UTF-8 --dir ~/cov-inter-netsniff-ng make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze netsniff-ng with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-netsniff-ng --all \
  --enable-callgraph-metrics

/* Commit defects to Coverity stream "netsniff-ng-mainline"
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream netsniff-ng-mainline --dir \
  ~/cov-inter-netsniff-ng --user username

/* ossec-hids -----------------------------------------------------*/

/* Configure and compile ossec-hids with Coverity */
$ cd ~/src
$ cp ~/Downloads/ossec-hids-2.8.2.tar.gz .
$ tar xvfz ossec-hids-2.8.2.tar.gz
```

```
$ cd ossec-hids-2.8.2/
$ cd src
$ cov-build --encoding UTF-8 --dir ~/cov-inter-ossec make all

/* Confirm Coverity returns "compilation units (100%)" then
   analyze ossec-hids with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-ossec --all --enable-callgraph-metrics

/* Commit defects to Coverity stream "ossec-mainline" */
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream ossec-mainline --dir ~/cov-inter-ossec --user username

/* snort ------------------------------------------------------------*/

/* Install required prerequisites for snort */
$ sudo apt-get install libpcap-dev libpcre3-dev libdumbnet-dev \
  bison flex
/* Install optional prerequisites (adds Adobe Flash support) */
$ sudo apt-get install liblzma-dev libnuma-dev

/* Configure and compile snort with Coverity */
$ cd ~/src
$ cp ~/Downloads/snort-2.9.8.0.tar.gz .
$ tar -xvzf snort-2.9.8.0.tar.gz
$ cd snort-2.9.8.0
$ ./configure
$ cov-build --encoding UTF-8 --dir ~/cov-inter-snort2/ make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze snort with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-snort2/ --all \
  --enable-callgraph-metrics
```

```
/* Commit defects to Coverity stream "snort2-mainline" */
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream snort2-mainline --dir ~/cov-inter-snort2/ --user username

/* Suricata ------------------------------------------------------------*/

/* Install prerequisites for Suricata */
$ sudo apt-get install libyaml-dev libcap-ng-dev libmagic-dev

/* Configure and compile Suricata with Coverity */
$ cd ~/src
$ cp ~/Downloads/suricata-3.0.tar.gz .
$ tar xvfz suricata-3.0.tar.gz
$ cd suricata-3.0/
$ ./configure
$ cov-build --encoding UTF-8 --dir ~/cov-inter-suricata make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze Suricata with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-suricata --all \
  --enable-callgraph-metrics

/* Commit defects to Coverity stream "suricata-mainline"
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects --host myserver.com --dataport 9090 \
  --stream suricata-mainline --dir ~/cov-inter-suricata --user username

/* Wireshark -----------------------------------------------------------*/

/* Install prerequisites for Wireshark */
$ sudo apt-get install qt-sdk libgtk-3-dev

/* Configure and compile Wireshark with Coverity */
$ cd ~/src
$ cp ~/Downloads/wireshark-2.0.1.tar.bz2 .
$ bunzip2 wireshark-2.0.1.tar.bz2
```

Jeff Sass, jsass@adobe.com

```
$ tar xvf wireshark-2.0.1.tar
$ cd wireshark-2.0.1/
$ ./configure
$ cov-build --encoding UTF-8 --dir ~/cov-inter-wireshark make

/* Confirm Coverity returns "compilation units (100%)" then
   analyze Wireshark with all of the Coverity checkers */
$ cov-analyze --dir ~/cov-inter-wireshark --all \
  --enable-callgraph-metrics

/* Commit defects to Coverity stream "wireshark-mainline"
   replace "username" with your Coverity username and "myserver.com"
   with your Coverity database server */
$ cov-commit-defects –host myserver.com --dataport 9090 \
  --stream wireshark-mainline --dir ~/cov-inter-wireshark \
  --user username
```