GRAMMATECH

A NEW ERA OF SOFTWARE FORENSICS WITH STATIC ANALYSIS

TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

The term “forensics” refers to the method of using science to discover evidence of criminal activity. Extending this to software broadens the use case to consider all the purposes of software investigation techniques. Software is ubiquitous, and is a part of all mission-critical systems. As such, software failures have tremendous real-world effects. In software, “forensics” refers to the method of using tools and techniques to uncover software evidence for purposes such as criminal investigations, civil cases (e.g. safety failures), commercial product failures, and security breaches.

BEYOND THE LAW

Investigating software mishaps serves multiple purposes. Though the approaches used may be similar across cases, the outcomes of the investigations are not necessarily associated with crime. A prime example would be the investigation of a software failure that has led to an accident resulting in injury, loss of life, or loss of property. Even if criminal activity or negligence is not suspected, investigators still approach the case using techniques similar to those used in a criminal investigation.

Software forensics encompasses any activity that requires analysis of source and binary code for investigation, post mortem analysis, or preventive measures. Some examples of use cases for software forensics include (but not limited to) the following:

- **Malicious code**: A common software forensics scenario would be detecting malicious code and tracking down its author. This code is written on purpose but with hidden intentions. Since detection using manual techniques can be difficult, manual inspections and regular software testing often fail to reveal the malicious code.

- **Safety incidents**: Software failures in safety-critical systems have significant impact on persons and property. Manufacturers are obliged to track down and investigate the root cause of these failures, either to settle civil suits or to research and prevent future incidents. For example, the FDA used CodeSonar to investigate the quality of various infusion pumps’ control software; and the NHTSB used CodeSonar to investigate Toyota’s unintended vehicle acceleration problem.

- **Security vulnerabilities**: Severe security breaches prompt an investigation into the root of the problem. Security vulnerabilities could be either intentional malicious code or accidental bugs in the software. Finding the root cause is critical for remediating these security vulnerabilities. NIST provides a security incident guide, which provides details on security vulnerability investigation and documentation techniques.

- **Software fault analysis**: In general, any software fault may be the subject of investigation. For example, a monitoring device providing inaccurate results could lead to overcharging a customer (household “smart meters,” for example). Courts expect due diligence in software development. Therefore, tools and techniques that are reasonable and commercially available should be used to investigate the cause of the software defect. Defects found in a fielded product that could have been prevented in development may be considered negligence and may point to culpbability of the developer.

THE ROLE OF STATIC ANALYSIS

Many approach software forensics operations using painstaking manual investigation of source and binary code. Detecting errors or traces of manipulation manually is difficult and time consuming. Automated tools and techniques, on the other hand, save time and money. Static analysis tools excel at detecting software errors and security vulnerabilities. They do so automatically and with high accuracy, and therefore play an important role in software forensics. Furthermore, static analysis forms an important part of the software development toolchain – building in security, safety, and quality ahead time avoids many costly downstream problems.

Malware, for example, is detected by advanced static analysis tools, which perform tainted data analysis from an input source (like a network connection) to where the data is used. If the destination or "user" of this data is a function with a security vulnerability, there is the potential for exploitation. GrammaTech's CodeSonar can detect security vulnerabilities in binary code, which is essential when source code is not available. Tainted data analysis can also determine if outside input can be used to manipulate a vulnerability into a full-fledged security threat. Automating this process saves huge amounts of time and effort that would have otherwise been spent on manual code analysis. Similarly, static analysis can accelerate other software investigations such as the research of a software safety incident to uncover software defects.

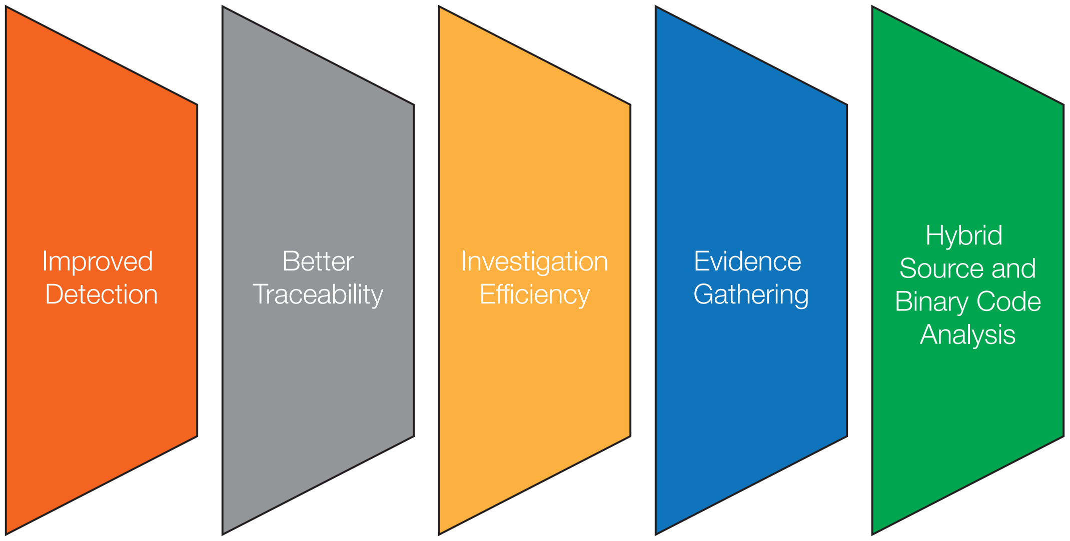


Figure 1: Benefits of advanced static analysis to software forensics.

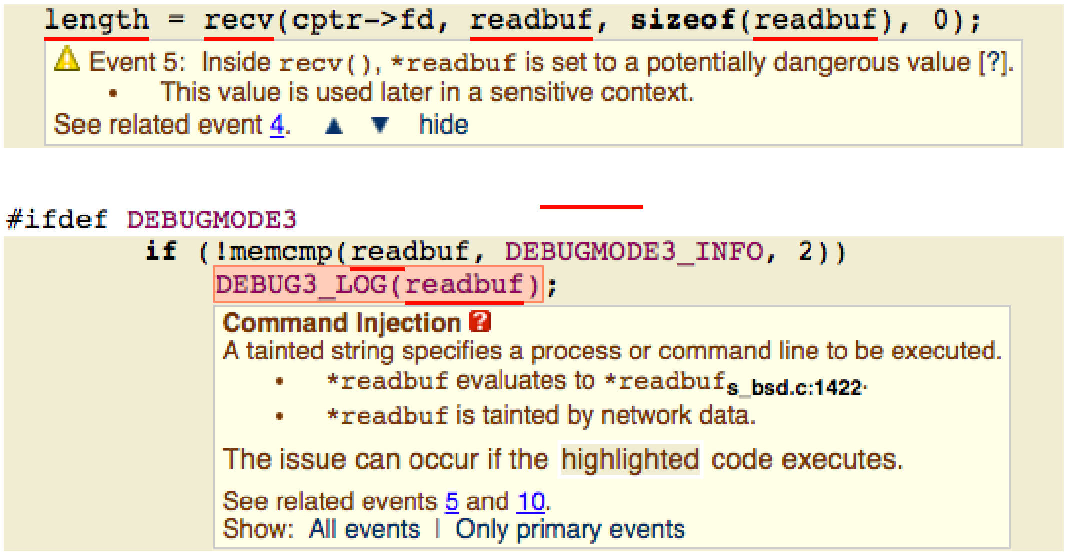
AUTOMATED BINARY CODE ANALYSIS FOR SOFTWARE FORENSICS

By automating something that is otherwise a painstaking job of manually reverse-engineering binary code, advanced static analysis provides many benefits, including the following:

- **Improved detection**: Automated static analysis can detect errors and vulnerabilities in binary executable or library object file of any size and across multiple compilation units and/or functions. The same analyses performed with manual approaches are difficult and often limited to pre-determined parts of the binary code. Although automated analysis may not entirely replace manual approaches, it certainly increases the detection rate and scope of analysis.
- **Better traceability**: One of the most difficult aspects of manual forensic investigation is tracing control and data flow. The difficulty is exacerbated when only binary code is available. Advanced static analysis performs control and data flow analysis on the entire scope of the application. This larger scope of analysis improves detection but also reduces false positives (errors reported that turn out to be false) and aids tainted data analysis. CodeSonar provides a sophisticated web user interface to significantly help with investigating errors.
- **Investigation efficiency**: Automated vulnerability detection, tracing, and reporting greatly increase the efficiency of binary code investigation. Software forensic investigators can rely on CodeSonar's binary analysis technology to quickly find vulnerabilities in the entire application. Meanwhile, manual investigation is time consuming and have to be focused on specific areas of the code. Considering the cost of manual binary code reverse-engineering and investigation, CodeSonar provides excellent return on investment.
- **Reporting to support evidence gathering**: Comprehensive error reporting, code visualization, and integration with vulnerability management systems are crucial to supporting a software security forensic investigation. Automating parts of the investigation-reporting means saving time and reducing errors.
- **Hybrid source and binary analysis**: Although this paper has focused thus far on binary code analysis, source code analysis is also critical if the source is available. For example, a disgruntled employee may leave a malware in source code "hiding in plain sight," which can be detected with an analysis of the source code. Automated binary code analysis shares the same features and benefits as those of automated source analysis. CodeSonar can report on both source and binary results in the same project. An example follows.

A SECURITY INCIDENT INVESTIGATION

A rather nasty exploit was discovered in the open source project Unreal IRCD as reported in CVE-2010-2075. In this case, external data received over a socket connection is used unchecked to perform system commands – a textbook command injection vulnerability. Figure 2 illustrates the source code in question in the `read_packet()` function.



```
length = recv(cptr->fd, readbuf, sizeof(readbuf), 0);
```

⚠ Event 5: Inside `recv()`, `*readbuf` is set to a potentially dangerous value [?].

- This value is used later in a sensitive context.

See related event 4. ▲ ▼ hide

```
#ifdef DEBUGMODE3
        if (!memcmp(readbuf, DEBUGMODE3_INFO, 2))
            DEBUG3_LOG(readbuf);
```

Command Injection

A tainted string specifies a process or command line to be executed.

- `*readbuf` evaluates to `*readbuf`s_bsd.c:1422.
- `*readbuf` is tainted by network data.

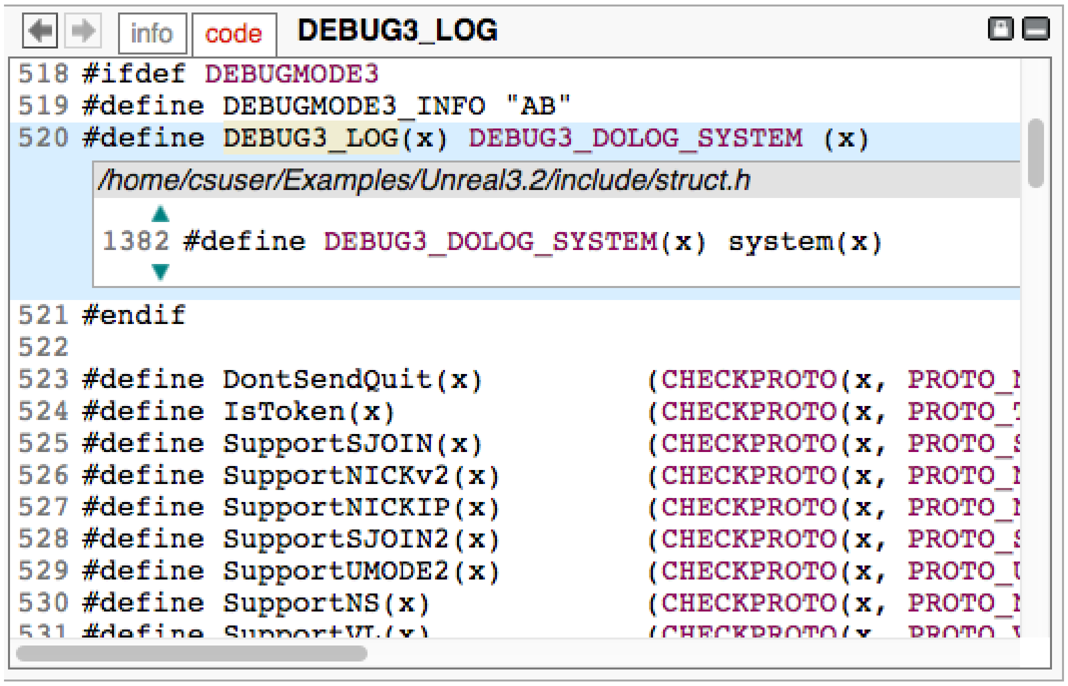
The issue can occur if the highlighted code executes.

See related events 5 and 10.

Show: All events | Only primary events

Figure 2: Code snipped from Unreal IRCD showing what appears to be a call to a harmless debug logging macro.

From first glance, this doesn't look like a problem. All that appears to be happening in this code snippet is debug logging – simply detecting an error and dumping the information to a log of some sort. However, the `DEBUG3_LOG` macro is purposely obfuscated and requires further investigation. In Figure 3 we can see that the definition of `DEBUG3_LOG` is a call to `system()`, which executes commands!



```
#ifdef DEBUGMODE3
#define DEBUGMODE3_INFO "AB"
#define DEBUG3_LOG(x) DEBUG3_DOLOG_SYSTEM (x)
```

/home/csuser/Examples/Unreal3.2/include/struct.h

```
1382 #define DEBUG3_DOLOG_SYSTEM(x) system(x)
```

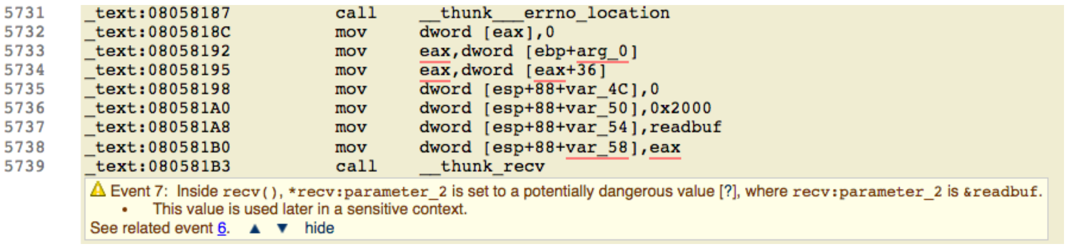
```
#endif

#define DontSendQuit(x)          (CHECKPROTO(x, PROTO_N
#define IsToken(x)               (CHECKPROTO(x, PROTO_T
#define SupportSJOIN(x)          (CHECKPROTO(x, PROTO_S
#define SupportNICKv2(x)         (CHECKPROTO(x, PROTO_N
#define SupportNICKIP(x)         (CHECKPROTO(x, PROTO_N
#define SupportSJOIN2(x)         (CHECKPROTO(x, PROTO_S
#define SupportUMODE2(x)         (CHECKPROTO(x, PROTO_U
#define SupportNS(x)             (CHECKPROTO(x, PROTO_N
```

Figure 3: CodeSonar provides quick access to macro expansions, which helps discover what `DEBUG3_LOG()` actually does.

Why does this look like an insider attack? This vulnerability was purposely written into the code and the macros have two levels of indirection in order to mask their real purpose. A hacker group claimed responsibility for this exploit and stated that it injected these macros into the download archive for Unreal IRCD.

ANALYZING THE UNREAL IRCD VULNERABILITY WITH BINARY ANALYSIS

Using CodeSonar's binary analysis, it's possible to detect security vulnerabilities from object, library, and executable code. This exploit is easier to spot with binary analysis. Figure 4 shows the warning from using a function known to read data from outside the process and how this data is being used in a call to `system(x)`. The compiler has completely stripped away any obfuscation in the source.

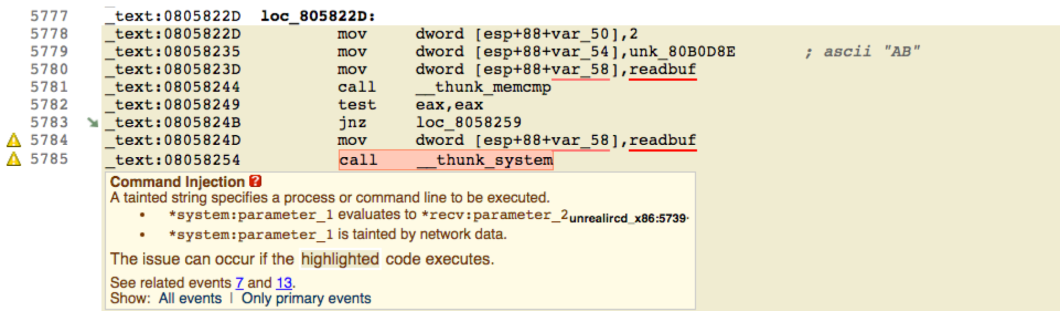


```
_text:08058187          call     __thunk___errno_location
_text:0805818C          mov      dword [eax],0
_text:08058192          mov      eax,dword [ebp+arg_0]
_text:08058195          mov      eax,dword [eax+36]
_text:08058198          mov      dword [esp+88+var_4C],0
_text:080581A0          mov      dword [esp+88+var_50],0x2000
_text:080581A8          mov      dword [esp+88+var_54],readbuf
_text:080581B0          mov      dword [esp+88+var_58],eax
_text:080581B3          call     __thunk_recv
```

Event 7: Inside `recv()`, `*recv:parameter_2` is set to a potentially dangerous value [?], where `recv:parameter_2` is `&readbuf`.

- This value is used later in a sensitive context.

See related event 6. ▲ ▼ hide



```
_text:0805822D  loc_805822D:
_text:0805822D          mov      dword [esp+88+var_50],2
_text:08058235          mov      dword [esp+88+var_54],unk_80B0D8E        ; ascii "AB"
_text:0805823D          mov      dword [esp+88+var_58],readbuf
_text:08058244          call     __thunk_memcmp
_text:08058249          test     eax,eax
_text:0805824B          jnz      loc_8058259
_text:0805824D          mov      dword [esp+88+var_58],readbuf
_text:08058254          call     __thunk_system
```

Command Injection

A tainted string specifies a process or command line to be executed.

- `*system:parameter_1` evaluates to `*recv:parameter_2` unrealircd_x86:5739
- `*system:parameter_1` is tainted by network data.

The issue can occur if the highlighted code executes.

See related events 7 and 13.
Show: All events | Only primary events

Figure 4: The same section of code as in Figure 2 analyzed by CodeSonar binary analysis. The vulnerability is much more obvious.

ADVANTAGES OF HYBRID SOURCE AND BINARY STATIC ANALYSIS

The Unreal IRCD vulnerability is a clear example of how static analysis, both for source and binary code, is used to investigate a real security incident. Binary analysis can detect errors that might be obfuscated in source or added maliciously after compilation. Below is a summary of the benefits of hybrid static analysis:

- **Two different views on the same vulnerability/defect**: Once the compiler has optimized the source code, the resulting compiled object code can reveal a different view of the detected errors. Although source is easier to read and understand, having two views on the same problem is better than one.

- **Detect injected code, modified binaries, and insider attacks**: A program's source is not its final state. Binary analysis can detect unwanted changes in the final executable. Code injected into executables or download payloads can be analyzed for defects and vulnerabilities. Malicious code added by inside attackers, possibly hidden in source, can be detected before shipping to customers.

- **Continue call graph into libraries and other binaries without source**: CodeSonar can analyze standard C/C++ libraries and any other third-party library or executable. When code makes calls to these libraries, the analysis continues in the binary realm. Analyzing both source and binary code means better detection and less false positives.

SUMMARY

Software forensics includes the investigation of source and binary code not only for the detection of criminal activity, but also for malicious code, safety software failures, and security incidents. In most cases, regardless of the purpose of the investigation, the techniques and tools used in software forensics are similar. Static analysis is a pivotal tool in software forensics because it increases the scope of analysis to the entire application and provides error detection with an accuracy difficult to duplicate with manual analysis. An example of how CodeSonar reduces the effort of manual binary code reverse-engineering was provided showing clear benefits in terms of security vulnerability detection and evidence gathering.

REFERENCES

"Federal Drug Administration (FDA) Recommends Static Analysis for Medical Devices", GrammaTech Case Study, 2014. https://www.grammatech.com/sites/default/files/grammatech-fda-case-study.pdf

"GrammaTech Used In Toyota Unintended Acceleration Investigation", GrammaTech Press Release, http://news.grammatech.com/grammatech-used-in-toyota-unintended-acceleration-investigation

"Computer Security Incident Handling Guide", National Institute of Standards and Technology, Special Publication 800-61 Rev 2. http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-61r2.pdf

"Vulnerability Details : CVE-2010-2075", CVE Details, 2010-06-15, http://www.cvedetails.com/cve/cve-2010-2075