GRAMMATECH

HOW STATIC ANALYSIS PROTECTS CRITICAL INFRASTRUCTURE FROM CYBER THREATS

TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

Flip a switch and a room is filled with light. Turn on the tap and clean drinking water flows. We are absolutely dependent on these marvels of modern infrastructure, and we take them completely for granted. It is only when the system breaks down that we are reminded of this reality. As we have seen in natural disasters, it is challenging to survive without power, clean water, food, sanitation, medicine, transportation, safety, and other conveniences of modern life. Fortunately, when disaster strikes in one region, areas that are untouched by the event can provide aid and assistance with recovery. What if, however, the US was the target of an orchestrated cyber–attack, intent on crippling the entire country? Films such as the National Geographic docudrama *American Blackout* paint a scary picture of such an attack. The real question is if this is just apocalyptic science fiction, or if it is actually possible for cyber terrorists to hack into our infrastructure and create widespread chaos.

According to Admiral Michael Rogers, head of the NSA and U.S. Cyber Command, "It's only a matter of the 'when,' not the 'if,' that we are going to see something dramatic." Reporting to a congressional panel, Rogers stated that several nation states have the capacity to shut down the nation's power grid and other critical infrastructure through a cyber attack. Over a dozen utilities have reported "daily," "constant," or "frequent" attempted cyber attacks on their systems – one utility reported 10,000 in one month, according to a 2013 congressional report on Electric Grid Vulnerability. According to a 2014 survey by ThreatTrack Security, 37% of energy companies have already been infiltrated. The cyber security firm FireEye has identified nearly 50 types of malware that specifically target energy providers. In many instances, cyber criminals used social engineering, sending emails that tricked workers into clicking on a bad link and installing malicious software.

At the core of critical infrastructure for power stations, water supplies, nuclear plants, chemical facilities, and transit systems is an industrial automation control system called Supervisory Control And Data Acquisition (SCADA). SCADA systems use real-time operational data collected from a distributed network of intelligent devices, sensors, and control outputs, to remotely monitor and control industrial processes from a centralized location.

SECURING SCADA SYSTEMS

Until recently, hackers generally didn't target SCADA systems because doing so required physical access, and critical infrastructure systems tended to be isolated from outside networks. Now, SCADA systems are widely connected to the internet, for convenience, big data analytics, and upgrades. In 2012, Project SHINE reported that 1,000,000 SCADA devices are connected to the Internet, with thousands being added daily. With the increasing network connectivity of these embedded devices, malicious hackers have new access through a virtual attack path. Unaware of this trend, embedded developers are largely uninformed of the risks of insecure code, focusing on safety and reliability versus the emerging cyber threats.

Given the high cost of failure and the exponential increase of cyber attacks, SCADA architects need to immediately address security end-to-end. This means starting with a "security-first" philosophy, knowing that there will be attacks. Developers of embedded and IoT systems need

to build in security and safeguards that are resistant to human error, natural disaster, and cyber attacks. Such security includes following a more robust development lifecycle process, adopting and mandating best practices, and using state of the art tools to ensure that all software being deployed is secure and robust.

One of the most useful, effective, and easy-to-use tools that development teams can utilize is advanced static analysis. Advanced static analysis engines implement comprehensive security checking, including taint analysis and exhaustive execution path analysis, far beyond what is humanly possible. It finds vulnerabilities and errors that would otherwise be missed during visual inspection, code reviews, and testing.

GrammaTech's CodeSonar thoroughly scans source and machine code, through advanced techniques, to identify bugs of all kinds, helping organizations improve code integrity by eliminating the security, safety, and reliability flaws from SCADA systems. CodeSonar identifies complex defects with easy-to-understand guidance, helping developers correct issues quickly based on a strong understanding of the seriousness of each defect they encounter.

TODAY'S REALITY

Software security vulnerabilities, including latent defects in embedded code, are being exploited by hackers to gain access to systems in order to subvert their use. These exploits are typically triggered when a hacker sends data over an input channel. Unfortunately, many systems today, including many SCADA systems, don't have processes in place to rigorously check input values. This leaves unchecked input values, which are defined as ***tainted*** – potentially opening the system to a hacker's control.

Tainted data values should always be a concern for developers, even when security is not an issue, because they can also cause system integrity issues, creating unexpected behavior and system crashes. Any software that reads input from any type of sensor should treat all values from the sensor as potentially dangerous. The values might be out-of-range due to a hardware failure. If the program is not prepared to check the values, then they might cause the software to crash later. The same techniques that defend against security vulnerabilities can also be used to defend against rogue data values. So, taint analysis should be considered as part of all organizations' development lifecycles.

Any software program can have a variety of inputs that are potentially hazardous. For example, a program that reads input from a device with an infra-red sensor should probably consider that channel to be dangerous. Security analysts define the points of exposure to a potential cyber attacker as the ***attack surface***. It is the sum total of the network attack surface, the software attack surface, and the physical attack surface. To assess a program's risk, it is useful to first gain an understanding of what its attack surface is, and this corresponds closely to the program's taint sources.

The electrical grid provides an extensive attack surface. The grid is actually a power delivery system made up of a complex network of power plants and transformers connected by more than 450,000 miles of high-voltage transmission lines. Power generated at power plants is

moved by transmission lines to substations. Local distribution systems then use smaller, lower-voltage transmission lines to move power from substations to consumers. Power system operation and control is managed centrally, using a SCADA enterprise software application. The electrical distribution system consists of regional substations that have multiple numbers of controllers, sensors, and operator-interface points. A potential attacker could gain entry through one or more of the many networked embedded devices that distribute power locally, or they could potentially breach the network through the SCADA enterprise application. Many embedded devices were put into service in the field decades ago, and were not designed for network connectivity. These devices at the edge of the network may be vulnerable to cyber attacks and pose a significant risk to the reliability of the nation's power grid.

Developers of embedded applications for power systems need to understand the attack surface of their software. One of the biggest risks is that an attacker will use an unverified channel to trigger a security vulnerability or cause the program to crash. Many types of issues can be triggered by a hacker taking advantage of tainted data, including buffer overruns, SQL injection, command injection, cross-site scripting, arithmetic overflow, and path traversal.

Programs take input from multiple sources, so the environment in which the program will execute determines the level of risk associated with each source. Taint sources include environment variables, file contents, file metadata such as a file's permissions or date stamps, the network, network services such as the results of a DNS query, the system clock, and the registry as found on Windows systems.

Tainted data vulnerabilities are notoriously difficult for developers to find because applications often use code from different sources. This creates unexpected attack surfaces that malicious hackers can exploit. Developers can identify and correct these security vulnerabilities much faster by using CodeSonar's visual taint analysis. Using sophisticated tainted data analysis, CodeSonar tracks potentially hazardous dataflows in C/C++ applications that are too complicated for developers to reliably find manually. Unlike other analysis tools that provide simple warnings for tainted values, CodeSonar uses its unique visualization engine to present vulnerabilities in a graphical format with an actionable and auditable interface.

The ability to visualize various properties of complex code can give developers important insights into software structure, behavior, and robustness. It provides a quick way to look at code and learn how it's organized and how it works. It can also help to pinpoint problem areas. CodeSonar's visualization features are designed to optimize visual inspection and analysis of software, offering real-time, fluid transitions between views at different levels of abstraction. CodeSonar doesn't just look at a single piece of software – it shows how the different components in a software system work together. When looking at machine code, visualization provides a unique advantage by helping developers get a quick picture of their code without digging into the semantics of the machine code.

EXPLOITING BUFFER OVERRUNS

Some of the most damaging cyber attacks in the last two decades have been caused by the infamous buffer overrun. As this is such a pervasive vulnerability, and because it illustrates the

importance of taint analysis, it is worth explaining in some detail.

There are several ways in which a buffer overrun can be exploited by an attacker. Here we describe the classic case that makes it possible for the attacker to hijack the process and force it to run arbitrary code. In this example, the buffer is on the stack. Consider the following code:

```
void config(void)
{
char buf[100];
int count;
…
strcpy(buf, getenv("CONFIG"));
…
}
```

In this example, the input from the outside world is through a call to getenv that retrieves the value of the environment variable named CONFIG.

The programmer who wrote this code was expecting the value of the environment variable to fit in the buffer, but there is nothing that checks that this is so. If the attacker has control over the value of that environment variable, then assigning a value whose length exceeds 100 will cause a buffer overrun to occur. Because buf is an automatic variable, which will be placed on the stack as part of the activation record for the procedure, any characters after the first 100 will be written to the parts of the program stack beyond the boundaries of buf. The variable named count may be overwritten (depending on how the compiler chose to allocate space on the stack). If so, then the value of that variable is under the control of the attacker.

This is bad enough, but the real prize for the attacker is that the stack contains the address to which the program will jump once it has finished executing the procedure. To exploit this vulnerability, the attacker can set the value of the variable to a specially-crafted string that encodes a return address of his choosing. When the CPU gets to the end of the function, it will return to that address instead of the address of the function's caller.

Now, if the code is executed in an environment where the attacker does not have control of the value of the environment variable, then it may be impossible to exploit this vulnerability. Nevertheless, the code is clearly very risky and remains a liability if left unfixed. Also important to note is that a programmer might also be tempted to re-use this code in a different program where it may not be safe to run.

This example is taking its input from the environment, but the code would be just as risky if the string was being read from another input source, such as the file system or a network channel. The most risky input channels are those over which an attacker has control.

MITIGATING RISKS WITH STATIC ANALYSIS

Static analysis tools have become very sophisticated over the years, and while early tools only

considered the behavior of individual statements and declarations, today's advanced tools can include the complete source code of a program to perform whole program analysis. These tools work much like compilers, taking source code as input, then parsing it and converting it to an Intermediate Representation (IR). Whereas a compiler would use the IR to generate object code, static analysis tools retain the IR which functions as a model of the program. The depth of the model determines the effectiveness of the tool. That depth is based on how much knowledge of program behavior is built in, how much of the program it can take into account at once, and how accurately it reflects actual program behavior.

Static analysis tools use checkers, which are software plugins that perform various types of analyses on the code. Checkers find defects and policy violations by traversing or querying the model, looking for particular properties or patterns that indicate problems. Using a symbolic execution engine, static analysis tools explore program paths, reasoning about program variables and how they relate. Advanced dataflow analysis prunes infeasible program paths from the exploration. When the path exploration notices an anomaly, a warning is generated. Static analysis is able to catch real bugs that are difficult to find by testing alone, including buffer overflows, memory leaks, use of uninitialized data, null pointer dereferences and concurrency violations.

The power of static analysis is that it does not rely on testing to find problems, nor does it require that an error or failure be reproduced. An advanced static analysis tool can infer the runtime behavior of a program without actually running the program. Furthermore, when it identifies a problem, it also pinpoints the locations within the code that created the failure. This makes the job of debugging far simpler. Static analysis does not eliminate the need for testing, but rather is complementary to it. The reality is that in large and complex software systems, there are so many possible state conditions and such an astronomical number of possible paths of execution, it is infeasible to exhaustively test them all. Static analysis, on the other hand, can explore these paths and state conditions, and is able to find problems that are missed by testing.

THE DEEPEST SOLUTION

CodeSonar offers the industry's most advanced static analysis tools and can detect more than a hundred types of problems right out of the box. Because it performs a unified dataflow and symbolic execution analysis that examines the entire program, without relying on pattern matching or similar approximations, it is able to find many types of bugs that are missed by other static analysis tools.

CodeSonar does not require changes to the source code or existing build system, and ideally should be run any time code is compiled. This makes it possible to find bugs as they are introduced, when the cost of fixing them is minimal. It is suitable for both small and large scale embedded software projects, and can perform whole-program analysis on more than 10 million lines of code. Once an initial baseline analysis has been performed, CodeSonar's incremental analysis capability makes it possible to quickly analyze daily changes to the code base.

An integrated development suite, CodeSonar helps automate team workflow and includes

powerful tools for program analysis, program inspection, program understanding, and architecture visualization. It enables developers to discover the underlying design intentions of existing code, and recognize when new code deviates from this design. It provides early warning when new defects are first introduced, and uses cutting-edge technologies to help developers identify and understand them.

CodeSonar identifies vulnerabilities that pose the biggest threat and thus helps protect against cyber attacks. It identifies and helps remove exploitable vulnerabilities quickly with a repeatable process. Additionally, it helps educate developers in secure coding practices while they work, and brings development and security teams together to find and fix security issues.

Embedded systems developed for modern infrastructure are not immune to cyber threats. Developers must take steps to ensure their systems are secure. This includes verifying all inputs to prevent tainted data from entering the system. It also includes eliminating any software vulnerabilities that could be exploited by hackers. Advanced static analysis tools help produce secure code by automating security best practices, including tainted data tracking. Equally important, static analysis plays an essential role in finding vulnerabilities and defects that could allow unexpected behavior that testing misses, and in helping developers understand and correct problems. When used in conjunction with other secure development lifecycle best practices, advanced static analysis tools can significantly reduce the risk of cyber attacks in critical infrastructure and other SCADA applications.

GrammaTech, Inc. is a leading developer of software-assurance tools and advanced cybersecurity solutions. GrammaTech helps organizations develop and release high quality software, free of harmful defects that cause system failures, enable data breaches, and increase corporate liabilities in today's connected world. GrammaTech's CodeSonar is used by embedded developers worldwide.