

# Using Static Analysis

---



**Dr. Jared DeMott**

SECURITY RESEARCHER AND ENGINEER

@jareddemott [www.vdalabs.com](http://www.vdalabs.com)



# Overview



**Introduce Automated Analysis**

**Pros vs. Cons**

**Review Cycle**

**Basic Usage of Tools**





## Check Code Automatically

- Development tools
- Security tools

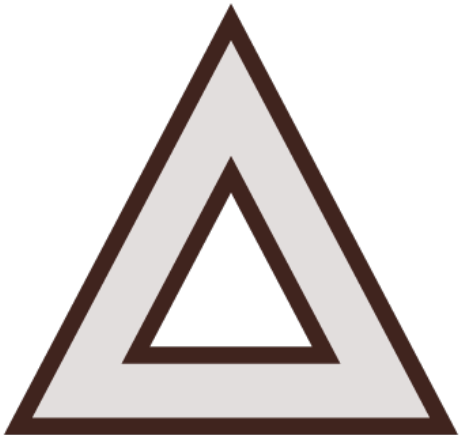


## Pros of Static Analysis Tools

- Runs across all code
- Extensible to find new bugs

## Cons of Static Analysis Tools

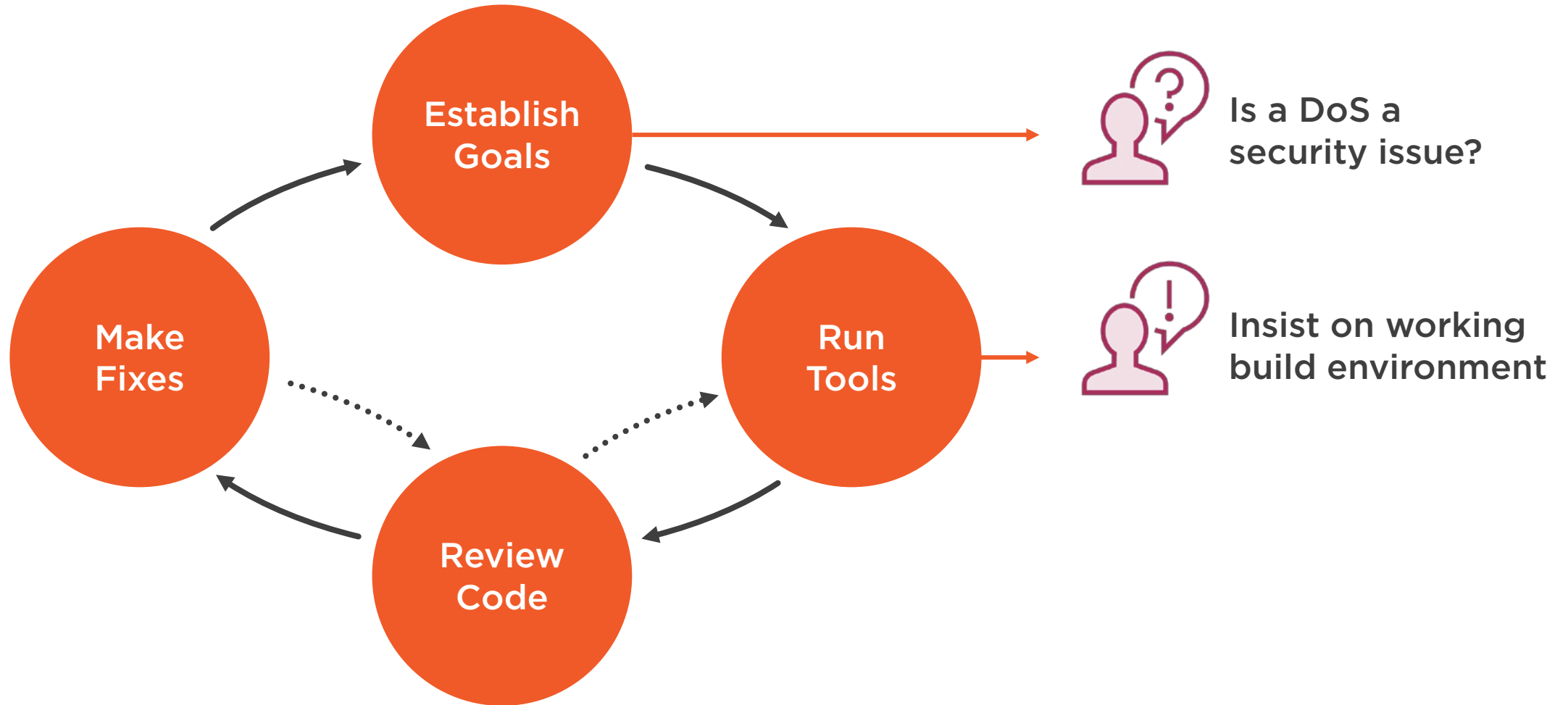
- FPs
- FNs



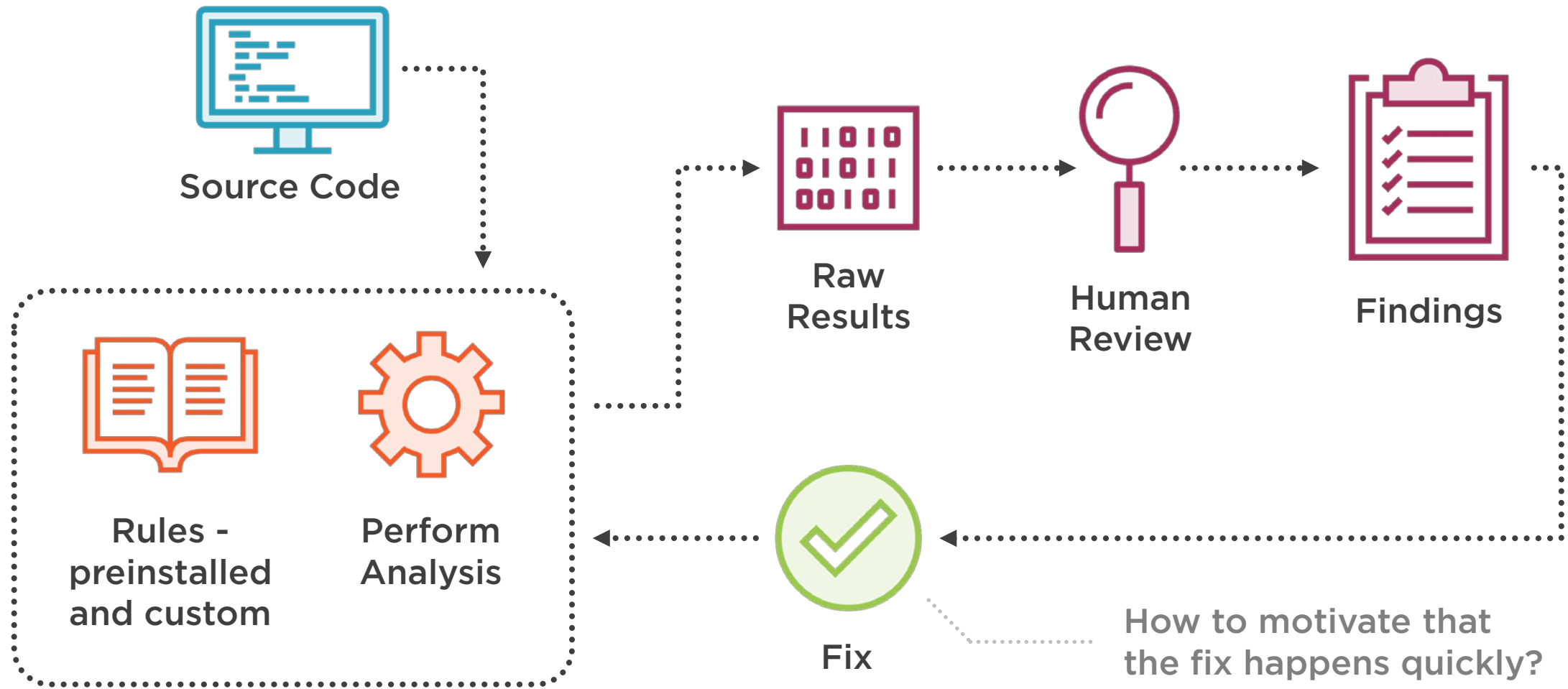
## When Using Automated Tools

- Roll them out slowly and carefully
- Central SAS dynamic testing

# Static Review Cycle



# Functional View of Static Analysis



# CppCheck

The screenshot shows the Cppcheck application window. The title bar is blue with the Cppcheck logo and standard window controls. The menu bar includes File, View, Program, and Help. The toolbar contains icons for running checks, saving, undo, redo, and help. A progress bar at the top right shows 6% completion. The main area is a table with columns for File, Severity, Line, and Message.

File	Severity	Line	Message
JavaScriptCore/runtime/JSString.cpp			
JavaScriptCore/runtime/GetterSetter.h			
JavaScriptCore/runtime/GetterSetter.cpp			
JavaScriptCore/runtime/JSNotAnObject.h			
JavaScriptCore/runtime/JSObject.cpp			
JavaScriptCore/runtime/RegExpConstructor.h			
JavaScriptCore/runtime/StringObject.cpp			
JavaScriptCore/wtf/dtoa.cpp			
JavaScriptCore/runtime/PropertySlot.h			
JavaScriptCore/API/JSCallbackObject.h			
JavaScriptCore/API/tests/minidom.c			
JavaScriptCore/API/tests/testapi.c			
JavaScriptCore/bytecode/CodeBlock.h			
JavaScriptCore/bytecode/SamplingTool.h			
JavaScriptCore/bytecode/SamplingTool.h	style	108	Redundant condition. It is safe to deallocate a NULL pointer
JavaScriptCore/bytecode/SamplingTool.h	style	278	The class 'AbstractSamplingCounter' has no constructor
JavaScriptCore/bytecode/SamplingTool.h	error	278	Class AbstractSamplingCounter which is inherited by class DeletableSampli...
JavaScriptCore/bytecode/SamplingTool.cpp			
JavaScriptCore/interpreter/Register.h			
JavaScriptCore/interpreter/Interpreter.cpp			
JavaScriptCore/jit/JITPropertyAccess.cpp			
JavaScriptCore/parser/ParserArena.h			



# Importance of Good SA Reports

Grouping and Sorting Results

Elimination Unwanted Results

Explaining the Significance of the Results

The screenshot displays the Fortify Static Analyzer interface. On the left, the 'Issues - (Fortify Default)' panel shows a summary of findings: 14 Hot issues, 45 Warning issues, 11 Info issues, and 70 All issues. Below this, a 'Group by: Category' dropdown is set to 'Category'. A list of issues is shown, including several 'Buffer Overflow (Data Flow)' errors in 'qwik-smtpd.c' at lines 374, 587, 590, 619, 620, and 627, and one 'Format String (Data Flow)' error at line 8.

The main window shows the source code for 'qwik-smtpd.c'. The following code snippet is highlighted:

```
583 {  
584     while((c = getc(config)) != EOF)  
585     {  
586         if(c == '\r' || c == '\n') break;  
587         line[i] = c;  
588         i++;  
589     }  
590     line[i] = '\0';  
591     fclose(config);  
592     return line;  
593 }  
594 }
```

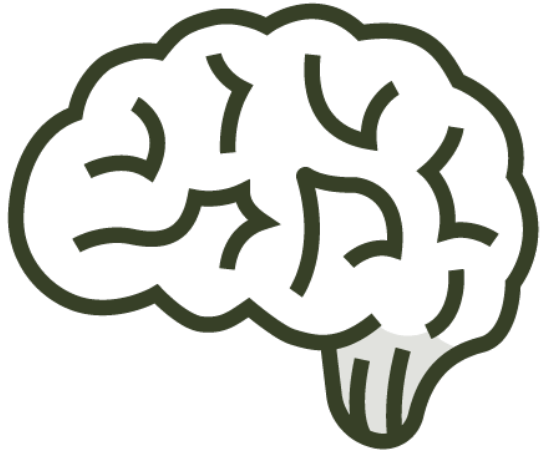
The bottom panel, 'Analysis Trace', shows the execution flow: 'getc(return) - qwik-smtpd.c:584', '[assignment to c] - qwik-smtpd.c:584', '[assignment to i] - qwik-smtpd.c:588', and 'assignment - qwik-smtpd.c:587'.

The 'Summary' tab is selected, showing an 'ABSTRACT' and an 'EXPLANATION' for the buffer overflow issue.

**ABSTRACT**  
Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

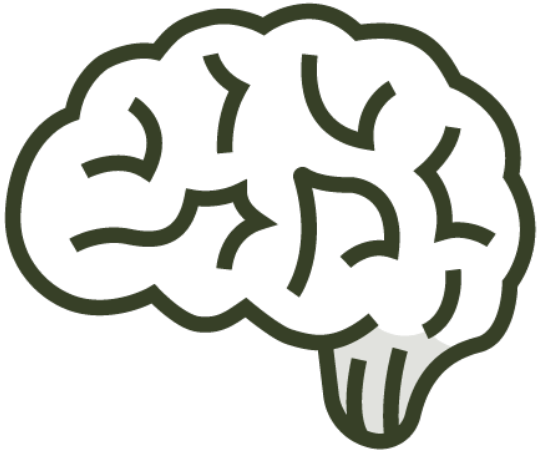
**EXPLANATION**  
Buffer overflow is probably the best known form of software security vulnerability. Most software developers know what a buffer overflow vulnerability is, but buffer overflow attacks against both legacy and newly-developed applications are still quite common. Part of the problem is due to the wide variety of ways buffer overflows can occur, and part is due to the error-prone techniques often used to prevent them.

In a classic buffer overflow exploit, the attacker sends data to a program, which it stores in an undersized stack buffer. The result is that information on the call stack is overwritten, including the function's return pointer. The data sets the value of the return pointer so that when the function returns, it transfers control to malicious code contained in the attacker's data.



## Prioritizing Reports is Important

- Particularly if just starting with static for the first time



## SA Tools

- Do not have a strong reputation for being useful to highly complex projects

# Summary



**Why Useful?**

**When to Include?**

**What's Next?**

