A Collaborative Marketplace for Continuous Software Assurance
A Unique Shared Facility

- Six proposals submitted
- Awarded to Morgridge Institute for Research with Indiana University, University of Illinois Urbana-Champaign, and UW–Madison as subcontractors
- Offers industry, academia and government agencies no-cost access to a secure research facility with analytical and reporting capabilities
- Will help the software assurance community improve the security of software used in the nation’s infrastructure
- Accelerate technology transfer to the market place by improving quality and functionality of SWA tools
Diverse Users

Software Developers

Upload software packages for analysis by a suite of software assurance tools and view results via dashboard.

Diverse Users

Software Assurance Tool Developers

Upload SWA tool and evaluate against large corpus of SW packages and test suites with known weaknesses.

Cybersecurity Researchers

Review data on tool coverage and common weaknesses to improve advance methodologies and techniques.
User Communities

- SWA Tool Developers
- SWA Researchers
- Software Developers
- Educators & Students
- Infrastructure Operators

Continuous Software Assurance Lab
Metronome

INPUTS
- Unvetted Software
- Software Assurance Research
- Software Assurance Tools
- Software Packages
- Identity Management
- User Training and Support
- Software Assurance Researchers
- Software Assurance Tool Developers

OUTPUTS
- State-of-the-Art Software Assurance
- Analysis and Baseline Repository
- Assessment Reference Data Sets
- Best Practices, False Positives
- Enhanced Cybersecurity
- Vetted Software
- Partner Resources
- U.S. Air Force
- National Security Agency
- U.S. Department of Homeland Security
- National Institute of Standards and Technology
- High Performance Computing Clusters

RESOURCES

SOFTWARE ASSURANCE MARKETPLACE
A NATIONAL CYBERSECURITY RESOURCE
Major Deliverables

- **SWAMP Operational** (Version 1.0 of CoSALab and Metronome)
- **V1 Stable Release of Metronome Second SWAMP User’s Meeting**
- **V2 of CoSALab and Metronome Third SWAMP User’s Meeting**
- **V3 of CoSALab and Metronome Third SWAMP User’s Meeting**
- **Final Metronome Release**

<table>
<thead>
<tr>
<th>Year</th>
<th>Phase</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Build</td>
<td>Oct. 1, 2012</td>
</tr>
<tr>
<td>2</td>
<td>Beta</td>
<td>Feb. 2, 2014</td>
</tr>
<tr>
<td>3</td>
<td>Enhance</td>
<td>Sep. 30, 2015</td>
</tr>
<tr>
<td>4</td>
<td>Operate</td>
<td>Sep. 30, 2017</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Oct. 1, 2013</td>
</tr>
</tbody>
</table>
Platform Selection for the SWAMP TBD

Guidelines:
- Minimally sufficient set. Initial DHS target is 8.
- Most relevant for SWA and OSS communities
- Updated current version and previous version

<table>
<thead>
<tr>
<th>Platform</th>
<th>Versions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux</td>
<td>Debian and/or derivatives</td>
</tr>
<tr>
<td></td>
<td>RedHat and/or derivatives</td>
</tr>
<tr>
<td></td>
<td>Others?</td>
</tr>
<tr>
<td>Mac?</td>
<td>OSX 10.8 (Lion), OSX 10.7 (Mtn Lion)</td>
</tr>
<tr>
<td>Windows</td>
<td>Windows7, Windows8</td>
</tr>
</tbody>
</table>
Initial Tool Selection for the SWAMP

At initial operating capability (IOC) a goal of 5 open source tools operating on 100 packages.

Guiding the selection of tools are weakness from published studies, communications with practitioners, and our experiences in performing in-depth software vulnerability assessments.

Weakness class examples include:

- Command injections
- SQL injections
- Use of inherently dangerous OS interfaces
- Buffer overruns
- Resource leaks (allocated but not freed resources)
- Pointer usage errors (e.g., NULL pointer usage, double freeing, use after free)
- Format string attacks
- Integer overflow/truncation errors
- Cross-site scripting/Cross-site request forgery
- URL redirection (Open Redirect)
# Initial Tool Selection for the SWAMP

<table>
<thead>
<tr>
<th>Tool Description</th>
<th>Languages</th>
<th>Targeted Weakness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Findbugs:</strong> widely used for Java source code analysis; incorporated into many commercial tools.</td>
<td>Java</td>
<td>Injection attacks, number handling, web deceptions, and resource leaks.</td>
</tr>
<tr>
<td><strong>cppcheck:</strong> community-wide open source tool hosted at sourceforce.net.</td>
<td>C, C++</td>
<td>Targets common coding errors, addressing buffer handling and resource leaks.</td>
</tr>
<tr>
<td><strong>Clang and Clang Static Analyzer:</strong> contributions from a wide community, including active participation from Apple and Google.</td>
<td>C, C++, Objective C (for MacOS)</td>
<td>Targets common coding errors, addressing injections, buffer handling, resource leaks, and number handling.</td>
</tr>
<tr>
<td><strong>Oink:</strong> based on CQual++ for its basic analysis. Reputed to be quite solid and has whole program analysis abilities.</td>
<td>C, C++</td>
<td>Targets common coding errors, addressing injections.</td>
</tr>
<tr>
<td><strong>PMD:</strong> a community wide tool under active development, hosted on sourceforge.net.</td>
<td>Java, limited support for XML, Javascript, JSP</td>
<td>Targets number problems, resource leaks, and programming errors. A focus on correctness, now adding security checks.</td>
</tr>
</tbody>
</table>
Ideal Static Assessment

- One Source, One Executable
- Source code
  - One file
  - Written in standards compliant dialect
  - No include files
- No compiler flags
- No libraries
- One tool
  - Source as input
  - Weaknesses as output
Real World Is More Complex

- Multiple source files
  - Separate compilation
  - Used to create libraries and executables
- Multiple executables
- Not obvious what to assess (no easily obtained list)
  - Generated files
  - System and 3rd party files outside of project directories
  - Command line arguments determine what is compiled, archived and linked, but also the source compiled through macro definitions and header file locations
- For each executable need to assess exactly those source files used in the creation of the executable
Difficulties in Determining Sources to Assess

- Build information is not declarative
- Program is run to build the software with many layers
  - Build configuration layer
  - Make drives build process
  - Calls shell code snippets
  - Generates source code and executables for build
- Capturing build tool invocations is non-trivial
Assessing a Package

- A package usually contains many executables
- Need to determine:
  - What executables exist
  - What source files used in build
  - Whole package analysis not acceptable
    Must be whole program by executable
- Too complex to leave to humans,
  requires automated tool
- Commercial tools seem to do whole package
  (all the sources compiled in the package)
Early Goal

- Generic open source tool to determine build information from arbitrary build process
- Capture executable invocation from build process
- Determine build information from known compiler, archiver, linker, and common utilities invocation
- This by itself would gives SwA tool developers a leg up
Build Monitor Tool Use

Run each tool once for each executable with discovered sources and compiler options, tool produces weaknesses, aggregate for viewing.
Community holds the key!

- We need your input – how do you envision using such a resource? What tools, packages, policies, topics, platforms would help you?
- We need your involvement – help with tools, packages, standards, technical literature, seminars, training.
- We need your feedback – the good, the bad, and the ugly.
Backup Slides
Multiple Executable per Package

- **Exec1 only Source**
  - Source1

- **Shared Source Between Exec1 and Exec2**
  - Source2
  - Source3
  - Source4

- **Exec2 only Sources**
  - Source5
  - Source6
  - Source7

- **Lib1**
  - Exec1

- **Lib2**
  - Exec2
Relationship to other DHS projects

Significant collaboration

TTA-1 Software Assurance Tools
TTA-14 Software Assurance Marketplace
Other Technical Topic Areas

Some collaboration
Software Assurance Marketplace Organization

~ 24 Team Members

Miron Livny

Chief Operations Officer
Brooklin Gore

Software Development
Production
Operations Center

User Support

Chief Security Officer
Von Welch

Security Operations
Indiana Univ. Pervasive Technology Institute

Chief Scientist
Barton Miller

Software Assurance Tools and Standards
U. of Wisconsin Middleware Security and Testing Group

Identity Mgmt. Lead
Jim Basney

External Resources
U. Of Illinois NCSA Cybersecurity Directorate

Morgridge Institute for Research