Introduction

• ONR office: Code 311 – Computer, Mathematics & Information Sciences Division)

• ONR Cyber Program Officers involved:
  – Mr. Gary Toth
  – Dr. Sukarno Mertoguno,
  – Dr. Ryan Craven
  – Dr. Daniel Koller
  – Mr. Ryan Gunst (clifford.gunst@navy.mil)

• All technical questions should be directed to Mr. Gunst
• All proposals should be addressed to Mr. Toth (1 May)
  – Detailed instructions in the BAA
• **Title:** *Late-stage Software Customization and Complexity Reduction S&T for Legacy Naval Systems*

• **Five Technical Areas (TA):**
  1. Functionality identification and reduction
  2. De-bloat/de-layer (functionality-preserving complexity reduction)
  3. Addition of security constructs
  4. Verification and Validation
  5. Supportive and complementary approaches

FY17
  - 28 Feb: BAA released
  - 14 Mar: Industry Day
  - 17 Apr: Deadline for questions
  - 1 May: Full proposals due by 1500 EDT

FY18
  - *16 Jun: Selection notification
  - *Q2FY18: Kickoff

* Approximate
Tech Push

• **Important:** This is not a *purely* basic research BAA

• Goal of this effort is to aggressively push novel and game-changing technology to the fleet

• In your proposals, consider how you will create **tools** that can be transitioned to and used by a government lab

• Univ.: this may require you to consider including engineering support

Types of S&T programs within ONR:

- **Discovery & Invention** (Basic and Applied Science)
- **Technology Push** (Leap Ahead, Innovative Novel Prototypes) ≈ 12%
- **Technology Pull** (FNCs, MinTech, TechSolutions) ≈ 30%
- **Quick Reaction S&T** (Scans, Works, Experimentation) ≈ 8%

Target customer

Current Fleet/Force

Fleet/Force in Development

Future Fleet/Force
BACKGROUND: Modern software is exceedingly complex and bloated

- Current practices encourage it (OOP, layers of abstraction, etc.)
- Priority is to maximize code reuse and increase programmer productivity
- One-size-fits-all feature set

Rube Goldberg machine

Complicated gadget that performs simple tasks in indirect, convoluted ways
BACKGROUND: Modern software is exceedingly complex and bloated
- Current practices encourage it (OOP, layers of abstraction, etc.)
- Priority is to maximize code reuse and increase programmer productivity
- One-size-fits-all feature set

- “In every application we looked at, an enormous amount of activity was executed to accomplish simple tasks.”
- “For example, a stock brokerage benchmark executes **268 method calls** and creates **70 new objects** just to move a **single date field** from SOAP to Java.”

Excerpted from:
Sevitsky et. al. (IBM TJ Watson Research Center) on framework based applications
http://lcsd05.cs.tamu.edu/papers/sevitsky_et_al.pdf
SW Complexity and Bloat

EXACERBATES SECURITY ISSUES:

- Widespread use of untrusted 3rd party libraries and runtime environments (broad attack surface)
- Security analysis is more difficult (more code, more complexity)
- One size fits all approach enables more robust attack planning and makes code reuse attacks easier

EXISTING APPROACHES:

- Patching
- Signature-based malware scanning
  
  REACTIVE

- API guards
- Shadow stacks
- Control-flow Integrity
  
  EXPENSIVE & OFTEN IMPERFECT

- Address Space Layout Randomization (ASLR)
- Stack canaries
- Anti-ROP (ret frequency counting, etc.)
  
  DEFEATABLE TRICKS

WEB BROWSER

Layers, wrappers

LIBRARIES (...)
- Webkit
- CSS Parser
- JavaScript Engine
- XML

RUNTIME

LIBRARIES (...)
- libXML
- libpng
- Openssl
- Etc...
**RESEARCH VISION:** Late-stage / install-time transformations

- Hard to change the way people write code, so work around it
- Series of automated transformations for legacy code
- Four independent, separate steps
  - Trimming tools should cut as aggressively as possible
  - Underlying analysis needed by each tool is different, but will likely overlap

![Diagram](image)
**RESEARCH VISION:** Late-stage / install-time transformations

- Hard to change the way people write code, so work around it
- Series of automated transformations for legacy code
- Four independent, separate steps
  - Trimming tools should cut as aggressively as possible
  - Underlying analysis needed by each tool is different, but will likely overlap

**Improving Software Robustness and Efficiency**

**Architecture & Strategy for Development & Deployment**

- **Feature Removal**
- **de-Layer de-Bloat**
- **Harden Security**
- **Verify and Validate**

Underlying analysis (code analysis, control-flow extraction, etc.)

**Executable code**
TA1: Functionality Identification and Reduction

- Functionality non-preserving; preserving with respect to the reduced set
- Someone needs to specify (admin?)
- Selecting desired features
- Feature removal
  - RE&A
  - Feature-code association
  - Communicating desired features
- Feature removal in dynamic languages
  - Specifying special order for dynamic language feature removal
TA2: De-bloat/de-layer

- Functionality-preserving
- Compiled and dynamic languages
- Reverse engineering & analysis (RE&A) of:
  - Binary
  - Bitcode, Bytecode, IR
  - Scripts
  - Undecidable $\leftrightarrow$ workaround w/ additional info
- de-Bloating / de-Layering
- Soundness and completeness
  - Dynamic can help, but what is practical?
  - Test-assisted? Need practical strategy for test deployment and execution...
  - Completeness is best effort
- What is the best strategy when a cut feature / function / etc. is invoked?
  - For feature removal $\rightarrow$ Throw error
  - For dLB $\rightarrow$ ??? Maybe error, or dynamically re-enable? Think about this.
  - Attacker invoked $\leftarrow$ DO NOT WANT!!!
TA3: Addition of Security Constructs

- It is unlikely, but security constructs may be trimmed out by earlier steps
- Security analysis & retrofitting
- RE&A
- Analysis with respect to common vulnerabilities and known exploits
- Beyond known stuffs: properties to assure
  - type-safety
  - others

![Cartoon Image of Security Constructs]

![Diagram of Security Processes]

COTS → Simplified COTS → Hardened COTS
• Functionality-preserving with respect to either full or reduced set of features
• Validation of functionality
• Verification of desired properties
• Formal assertions of (security) properties
  – Formal model of execution environment
  – Extracted formal model of the program/application
  – Formal specification of properties to assure
• **Feature Removal**
  – Cut unneeded functionality (admin-assist)
  – Is a functionality-preserving transformation but only for **desired** features

• **Complexity Reduction (dLB)**
  – Functionality-preserving transformation for the aggressive reduction of code size/complexity, indirection, and layers of abstraction

• **Retrofitting Security**
  – Security-focused code analysis and functionality-preserving transformations for enhancing robustness and security

• **Asserting correctness and security**
  – Automated and *in situ* verification of validation to ensure the transformation results are robust and secure
TA5: Supportive and complementary approaches

• Intended to catch research complementary to TA1-4
• Enabling or cross-cutting work that will improve the accuracy or robustness of TA1-4 approaches
  – Robust executable or binary reverse engineering tools
  – Robust binary -> intermediate representation (IR) transforms
  – Methods to improve robust IR extraction
  – Automated fundamental software transformations that improve the quality & simplicity of software or reduce attack surface of systems and software
    • REMEMBER OUR FOCUS ON LEGACY
    • Nothing that requires being in a VM, no IDS stuff
Advice

We recommend proposals consider the following:

• Must work without developer cooperation (**legacy**)!
  – May not have source code available

• Be very cognizant of your approach's limitations
  – Binary transformation is unsolved & very difficult
  – Over-confidence without supportive evidence will not instill confidence in your approach amongst the reviewers
  – Some limitations are fundamental (e.g., undecidability), but that’s OK—we can still make lots of progress for practical applications
  – Just need to be aware of our limits and clearly define them

• The less system properties you rely on, the better
  – Approaches that require VM introspection, etc. are discouraged
  – Static transformations are preferred

• Automation, automation, automation!
  – Semi-automated tools that aid a human are OK (e.g., an admin doing feature cutting)
We recommend proposals consider the following:

• Define your scope well:
  – Compiled or interpreted languages? Or both?
  – Which layer? Web, Apps, OS, Hypervisor, BIOS/firmware?
  – None has the advantage over the other, just be upfront. Lots of stuff is used throughout the Navy

• Think about how to avoid clashing with current development methodologies (adds constraints for you, but reduces deployment burden in the end)

• This is cool stuff! Have fun :)
Programmatics

- **Award lengths:**
  - Requesting 1-3 year base + options (total NTE 5 years)
  - Anticipated start dates:
    - Grants: Nov 1, 2017
    - Contracts: Jan 1, 2018

- **Funding levels:** Up to ~$1.5M/yr (smaller efforts are OK)

- **IMPORTANT:** Recall tech push slide...need to deliver tools
  - Options can be for maturing tools, transition work w/ Navy, etc.
  - Later years may involve some collaborative and integrated research with naval environment for deliverables
  - Expect to (at minimum annually) spiral code out to our ‘tool curator’
  - Tool curator will help us with maturation, QA, documentation & training, etc.
Oct 2016 Workshop:
- A public, open audience workshop was hosted by the ACM 2016 International Conference on Communications and Computer Security (ACM CCS 2016)
- Workshop was titled **FEAST: Forming an Ecosystem Around Software Transformation**
- ONR POs were involved in guiding the focus of the workshop, which examined various issues and limitations surrounding software customization
- We recommend taking a look at the slides for the keynote which helped frame discussion
- Plug: FEAST will likely continue at CCS 2017 in Dallas, TX

Goals of the workshop:
- Explore issues in software/feature reduction
- Study software development & deployment strategies for enhancing efficiencies
- Discuss how to retrofit security in compiled legacy code when source code is and is not available
- Explore current state-of-the-art techniques for supporting late stage software customization (e.g., binary analysis, reassemblers, etc.)

https://sites.google.com/site/ccsfeast16/ or, just google for “ONR FEAST”
FAQ: How would I apply patches?

**STRATEGY:** Recall that transformations are at install-time

- SW developers keep same workflow
- Vanilla copy is kept (potentially on system) when system owner first applies automated tools
- Patches are applied to vanilla copy; tools must be rerun

Maintain (potentially on system):

Vendor patch

Vendor-supplied code

Transformed code / shadow libraries

Execute:

Furthers need for the tools to be automated