Assured Micropatching (AMP)

Sergey Bratus
DARPA
Information Innovation Office (I2O)

Briefing prepared for Proposers Day

26 September 2019

Images of specific products throughout this presentation are used for illustrative purposes only. Use of these images is not meant to imply either endorsement or vulnerability of a product or company.
Rapidly produce targeted security patches (micropatches) to repair legacy binaries of mission-critical systems, with strong guarantees that the patch will not impact the functions of the system.
Patching known vulnerabilities is hard

GAO 2018 report examples:
- Only 1 in 20 vulnerabilities found by a prior audit was patched
- Known flaws unpatched for years

Patching a known flaw in a binary is hard:
- Platform options and optimizations diverge exponentially
  - GCC has 200+ parameters that affect compilation
- Build processes are custom; build chains are brittle, restrictively licensed
- 3rd party dependencies are ubiquitous
- Embedded SDKs lag behind stock versions of libraries in versions and fixes

Deployed binaries, each with a differently built library binary unit

Will rebuilt binaries re-integrate and pass tests?
Challenge: Rebuilding and re-integration needs costly manual effort

Desired: Patched binary with minimal changes, successfully integrates

Integratable binaries

Rebuilt binary, fails to integrate

Original source code

Patched code

Build process

Binary

Actual binary

New build process

Original build process
Challenge: Rebuilding and re-integration needs costly manual effort

Desired: Patched binary with minimal changes, successfully integrates

Semantic equivalent, but fails to situate the patch

Source code that can situate the patch

Original source code

Patched code

Integratable binaries

Actual binary

New build process

Original build process

Manual

Build process

Source

Binary

Binary analysis

Rebuilt binary, fails to integrate

DISTRIBUTION STATEMENT A - Approved for public release; distribution is unlimited.
Vision: Micropatches with certification evidence via automation

- Original source code
- Patched code
- Assured recompilation with guarantees of non-interference

- Goal-driven decompilation
- Automated

- Actual binary
- Integratable binaries
- Patched binary with certification evidence

- Semantically equivalent, but fails to situate the patch

DISTRIBUTION STATEMENT A - Approved for public release; distribution is unlimited.
Vision: Micropatches with certification evidence via automation

- **TA1**: Goal-driven decompilation
- **TA2**: Assured recompilation with guarantees of non-interference
- **TA3**: Evaluation

Semantically equivalent, but fails to situate the patch

Original source code

Integratable binaries

Patched binary with certification evidence

**Source**

**Binary**

**Actual binary**

**Patched binary with certification evidence**

**DISTRIBUTION STATEMENT A - Approved for public release; distribution is unlimited.**
### Vision: Assured security patches in hours, not months to years

#### Patching workflow

<table>
<thead>
<tr>
<th></th>
<th>Today: ≥ 6 months to years</th>
<th>Tomorrow: hours to days</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Decompilation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Recover source code</td>
<td>Manual, iterative decompilation</td>
<td>Goal-driven decompilation</td>
</tr>
<tr>
<td>Produce and situate the patch</td>
<td>Manual analysis and code adjustment</td>
<td>Targeted automated patch generation and placement</td>
</tr>
<tr>
<td><strong>Recompilation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compile patched code</td>
<td>From-scratch compilation (likely introduces flaws, destroys prior integration)</td>
<td>Assured recompilation Non-interference analysis and proofs</td>
</tr>
<tr>
<td>Re-integrate compiled binary module</td>
<td>Manual placement and re-linking</td>
<td>Automated relinking mindful of constraints</td>
</tr>
<tr>
<td>Testing</td>
<td>Full scope re-testing</td>
<td>Proof-guided, focused re-testing</td>
</tr>
<tr>
<td>Certification evidence</td>
<td>Dependent on full re-testing</td>
<td>Informed by non-interference proofs</td>
</tr>
</tbody>
</table>

#### Certification Deployment

- **Manual**
- **Automated**
TA1: Goal-driven decompilation

Challenges:
• Search the space of semantically equivalent representations of a program subject to relevant ideas of equivalence explicitly specified or embodied in a tool
• Goal-driven decompilation, to produce a decompiled representation of a binary unit best fit for the task
  • Goal functions: similarity, descriptional complexity

Challenge problems:
• Decompile binary to most closely approximate given program style, structure
• Decompile binary to achieve most succinct descriptions of functionality
  • Recover concise bytecode semantics of a virtual machine
  • Recover concise mathematical formulas used by a control process
• Decompile binary to optimize performance of analysis tools

Potential approaches:
• Rich, adaptable theories of semantic equivalence
• Adaptive mixing of emulation, symbolic/concolic execution, tracing, etc.
• Open architectural framework of composable modules (cf. LLVM)
Challenges:
• Precisely track the effects of patches from appropriate decompiled or intermediate representation to binary throughout (re)compilation, identify the footprint of changes on unit tests, rigorously verify non-interference
• Recover or approximate program units and interfaces in families of Intermediate Representations (IRs)
• Recover or approximate build processes to the extent suitable for producing assured micropatches

Challenge problems:
• Instrument rebuild process with symbolic tracking of the change footprint
  • Firmware modules: Automotive Engine Control Unit (ECU)
  • Standard library (OpenSSL), Linux Kernel Module (LKM)/driver

Potential approaches:
• Machine learning of assembly and compilation
• Well-structured succession of program analyses
TA3: Exemplary evaluation use case: heavy vehicle domain firmware

Provide tests of increasing difficulty culminating in networked system

Phase 1: Commodity system
Telematics unit

Phase 2: Real-time system
Engine/Brake Control Module (ECU)

Phase 3: Networked system
Truck-on-a-board testbed

DISTRIBUTION STATEMENT A - Approved for public release; distribution is unlimited.
### Evaluation metrics and milestones

<table>
<thead>
<tr>
<th></th>
<th>Phase 1 (18 months) Commodity</th>
<th>Phase 2 (18 months) Real-time</th>
<th>Phase 3 (12 months) Networked system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exemplary platform for a heavy vehicle use case</td>
<td>Telematics/ELD unit (commodity OS router/IoT device firmware)</td>
<td>Single ECU firmware (engine, brakes control modules)</td>
<td>Heavy vehicle ECU as a system (truck-on-a-board)</td>
</tr>
<tr>
<td>Testbed platform challenge, size and complexity</td>
<td>Commodity connected device, 100k LOCs, 5-10 components</td>
<td>Real-time control device, 1M LOCs, 10-50 components</td>
<td>Multiple connected devices of 1M+ LOCs</td>
</tr>
<tr>
<td>TA1: Goal-driven decompilation</td>
<td>Human-in-the-loop decompilation of individual functions at 60% accuracy relative to ground truth</td>
<td>Automated decompilation of modules at 80% accuracy relative to ground truth</td>
<td>Automated decompilation integrated with industry tools, of realistic binaries at 95% accuracy</td>
</tr>
<tr>
<td>TA2: Assured recompilation</td>
<td>Unoptimized compilation of patched code for a single library CVE, statically linked, with no loss of functionality vs ground truth; Semi-manual proofs in 1-2 weeks</td>
<td>Optimizing compilation of patches with dynamic linking and multi-procedure analysis, with no loss of functionality vs ground truth; Automated proofs 1-2 days</td>
<td>Full build chain recovery, integration with industry tools; Fully automatic proofs Hours</td>
</tr>
</tbody>
</table>

**Legend:**
- CVE: Common Vulnerabilities and Exposures
- ECU: Electronic Control Unit
- ELD: Electronic Logging Device
- IoT: Internet of Things
- LOCs: Lines of Code
- OS: Operating System
• Each performer participates in technical challenge events
  • Government reserves the right to engage third parties to independently validate results

• TA3 performer responsible for specifying in their proposal which devices they will use, for each of the phases
  • Make your choices based on proposed phased goals
  • Suggestion: Specify groups of devices in each phase
    • Government may choose to limit to a subset, or propose substitute devices during contract negotiations
### Program Schedule

<table>
<thead>
<tr>
<th>TA1: Goal-driven decompilation</th>
<th>Phase 1</th>
<th>Phase 2</th>
<th>Phase 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Commodity</td>
<td>Real-time</td>
<td>Networked system</td>
</tr>
<tr>
<td></td>
<td>18 months</td>
<td>18 months</td>
<td>12 months</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TA2: Assured recompilation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demonstrations</td>
</tr>
<tr>
<td>Tests</td>
</tr>
</tbody>
</table>

DISTRIBUTION STATEMENT A - Approved for public release; distribution is unlimited.
Meetings and Reporting Requirements

- Two annual Principal Investigator (PI) meetings centered around a challenge problem
- PM site visits between PI meetings
- Monthly financial reporting
- Quarterly technical progress reporting
  - Technical report describing progress, resources expended and issues requiring Government attention, provided 10 days after the end of each quarter
- System Development Plan provided one month after the kick-off meeting for each phase
  - Describe the scope/design and hardware and software architecture
- Real-time reporting of publications, prototypes, significant results via website/wiki or similar mechanism
- Financial/technical progress reporting to DARPA Technology Financial Information Management System (TFIMS)
- Final Technical Report
Funding and Programmatic Details

• Proposals due: Wednesday, November 20, 2019 at 12:00 noon (ET)
• Government anticipates multiple awards
  • Procurement contracts, cooperative agreements or other transactions
• Organizations can submit separate proposals to all Technical Areas (TAs)
  • Each proposal may address only one TA
  • Which to consider for award is at the discretion of the Government
  • A TA3 submission excludes a TA1 and TA2 award
• To expedite award contracting, proposers are encouraged to have sub-award agreements in place ahead of award notification
Q & A

• Does AMP assume that a comprehensive test suite for the target binaries is available?
  • No. However, some tests may be available, and strong proposals will discuss ways to take advantage of them for building the strongest possible assurance evidence.

• Does AMP assume that a functional specification for the target binaries is available?
  • No. In the most common case, the existing binary constitutes the ground truth on the certified functionality.

• Does AMP assume that full source code used to create the target binary is available?
  • No. However, some samples of such source code or related source code may be available, and strong proposals will discuss ways to take advantage of them.

• Does AMP assume that the original build process used to create the target binaries is available?
  • No. However, some parts or artifacts of the build process may be available, and strong proposals will discuss ways to take advantage of them.

• What baseline case should the tools developed by TA1 and TA2 address by the end of Phase I?
  • The firmware image of a connected device is known to contain a vulnerable library exposed to hostile inputs. A source-level patch for a stock version of this library is available, however it is not known whether the SDK version of the library used to create the firmware is the same as the stock version. A version of the compiler for the device’s platform is available but it is not known whether this exact version was used to create the firmware. A subset of functional requirements for the device are available, are fulfilled by the existing firmware, and must be preserved in the patched firmware.
• Does “micropatching” mean producing the smallest possible bit-wise change?
  • No. From the AMP perspective, the most important property of the binary patch is the depth of analysis and hence the level of assurance that could be obtained about it, not the size.

• Do the envisioned tools allow human-in-the-loop?
  • Yes. Although automation is preferred wherever possible, the key goal of AMP is to reduce the time needed for producing assured micropatches from month to days or hours. This goal does not preclude the human-in-the-loop operator as such.

• What level of competence is assumed about the human operator of the created tools?
  • The human operator is assumed to have a programmer’s understanding of the mitigation approach for the vulnerability, such as the source-level patch. The operator is not assumed to have either reverse-engineering or program verification skills.

• Is the program scope limited to any particular kinds of static or dynamic program analysis?
  • No. Any and all kinds and combinations of program analysis techniques are in scope, so long as they serve the purposes of the program: rapid security micropatching of mission-critical systems with strong guarantees of non-interference with baseline functionality.

• Do tools created by the different TA1 and TA2 performers need to interoperate?
  • Yes. It is expected that goal-driven decompilers created by TA1 will be usable as stand-alone tools, and will interoperate with the TA2’s recompilers and binary difference analysis tools via well-defined mutually intelligible Intermediate Representations (IRs).