Advanced Anti-Deobfuscation

Bjorn De Sutter

ISSISP 2017 – Paris
About me

• Research domain: system software
  • compilers, binary rewriting tools, whole program optimization (binary & Java), virtualization, run-time environments
  • improve programmer productivity by means of automation
  • apply tools for different applications
    • obfuscation, diversity, mitigating side channels and fault injection, ...
    • protect against exploitation of vulnerabilities (multi-variant execution)
    • generating code for accelerators

• Also worked/spent time at

• Interrupts enabled
About me

ASPIRE Framework

Decision Support System

Software Protection Tool Chain

Protected SafeNet use case

Protected Gemalto use case

Protected Nagravision use case

Data Hiding Algorithm Hiding Anti-Tampering Remote Attestation Renewability

http://www.aspire-fp7.eu
Lecture Overview

1. Basic Attacks
   • attacks on what?
   • basic attack tools & techniques

2. Defenses
   • anti-anything

3. Advanced Automated Attacks
   • generic deobfuscation
   • symbolic execution

4. Defenses
   • anti-even-more
<table>
<thead>
<tr>
<th>Asset category</th>
<th>Security Requirements</th>
<th>Examples of threats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Private data</strong> (keys, credentials, tokens, private info)</td>
<td>Confidentiality, Privacy, Integrity</td>
<td>Impersonation, illegitimate authorization, Leaking sensitive data, Forging licenses</td>
</tr>
<tr>
<td><strong>Public data</strong> (keys, service info)</td>
<td>Integrity</td>
<td>Forging licenses</td>
</tr>
<tr>
<td><strong>Unique data</strong> (tokens, keys, used IDs)</td>
<td>Confidentiality, Integrity</td>
<td>Impersonation, Service disruption, illegitimate access</td>
</tr>
<tr>
<td><strong>Global data</strong> (crypto &amp; app bootstrap keys)</td>
<td>Confidentiality, Integrity</td>
<td>Build emulators, Circumvent authentication verification</td>
</tr>
<tr>
<td><strong>Traceable data/code</strong> (Watermarks, finger-prints, traceable keys)</td>
<td>Non-repudiation</td>
<td>Make identification impossible</td>
</tr>
<tr>
<td><strong>Code</strong> (algorithms, protocols, security libs)</td>
<td>Confidentiality</td>
<td>Reverse engineering</td>
</tr>
<tr>
<td><strong>Application execution</strong> (license checks &amp; limitations, authentication &amp; integrity verification, protocols)</td>
<td>Execution correctness, Integrity</td>
<td>Circumvent security features (DRM), Out-of-context use, violating license terms</td>
</tr>
</tbody>
</table>
What is being attacked?

1. Attackers aim for assets, layered protections are only obstacles
2. Attackers need to find assets (by iteratively zooming in)
3. Attackers need tools & techniques to build a program representation, to analyze, and to extract features
4. Attackers iteratively build strategy based on experience and confirmed and revised assumptions, incl. on path of least resistance
5. Attackers can undo, circumvent, or overcome protections with or without tampering with the code
Basic Attack Techniques

• **Static** attack steps: without executing the code
  • symbolic information
  • graph representations of program

• **Dynamic** attack steps: observing execution
  • all kinds of hooks
  • start and intervene at interfaces
  • observe features and patterns of program execution (traces)

• **Hybrid** attack steps: combination of both
  • e.g.: build graphs of (unpacked) code observed during execution
Disassemblers - 1

• IDA Pro
• Binary Ninja
• angr

• Far from perfect
  • incomplete disassembly
  • incorrect graphs (control flow, call graphs)

• Flexible and interactive
  • linear sweep, recursive descent, heuristical and manual disassembly
  • GUI
  • code annotation
  • plug-ins and scripts
Disassemblers - 1

IDA Pro
- Binary Ninja
- Far from perfect
- Incomplete disassembly
- Incorrect graphs (control flow, call graphs)
- Flexible and interactive
- GUI
- Annotation
- Plug-ins and scripts
Disassemblers - 2

- Static & hybrid attacks
- Rely on many underlying assumptions
- Library detection
  - F.L.I.R.T
- Diffing tools
  - BinDiff
- Custom tools
  - detect patterns
  - undo obfuscations
  - data flow analysis
- Supports code editing
- Interfaces with (remote) debuggers
Disassemblers - 2

- Static & hybrid attacks
- Library detection
- F.L.I.R.T
- Diffing tools
- BinDiff
- Custom tools
- detect patterns
- undo obfuscations
- data flow analysis
Disassemblers - 2
Disassemblers - 3

• Decompiler
Debuggers - 1

• GDB
• OllyDbg

• Scriptable

• Support tampering
  • alter processor state (incl. program counter)
  • alter memory contents
  • alter code
  • used for out-of-context execution
Debuggers

- GDB
- OllyDbg
- Scriptable
- Used for tampering
  - alter processor state (incl. program counter)
  - alter memory contents
  - alter code
- used for out-of-context execution
Debuggers - 2

• Used for program understanding
• Used for zooming in on relevant code
  • Continuous iterative refinement of scripts
• Low overhead with hardware breakpoints
• High overhead with software breakpoints
  • Requires tampering
Emulation & Instrumentation

- QEMU
- Pin
- Valgrind
- DynInst
- ltrace

- Used to collect traces
  - To identify patterns and points of interest
- Used like a debugger
  - Iterative refinement of scripts
  - But not interactive
Software Tampering

• Editing the binary
• Alter running process state (CPU, memory)
• Intervene at interfaces
  • system calls
  • library calls
  • network activities
  • ....
• Custom binaries to invoke library APIs
• Aforementioned tools
• Cheat Engine
  • all kinds of reverse engineering aids (pointer chaining)
Pointer chaining

```
struct player

bool visible
```
Pointer chaining

```
struct player
    bool visible
```
Pointer chaining

*(ESP(play())-0x16)+0x4)+0x28
Pointer chaining
Lecture Overview

1. Basic Attacks
   • attacks on what?
   • basic attack tools & techniques

2. Defenses
   • antianything

3. Advanced Automated Attacks
   • generic deobfuscation
   • symbolic execution

4. Defenses
   • anti-even-more
Anti-tampering

• Code guards (code integrity)
  • hashes over code regions

• State inspection
  • check for existing invariants
  • inject additional invariants
  • for data integrity and control flow integrity

• Basic control flow integrity
  • check return addresses
  • check stack frames
Remote attestation

---

### Attestation Process

1. **Attestator Routine**
   - **Details**: An attestation routine is invoked that returns the attestation report in the form of some data. It should be noted that the attestation routine can be a single monolithic routine, but it can also consist of a collection of smaller routines that are invoked one after the other to iteratively compute an attestation report. In the latter case, the routines can actually also be in-lined into the program and hence be indistinguishable from the application code.

2. **Verifier Routine**
   - **Details**: In step 2, which will typically be executed immediately after step 1, the attestation report is verified. This can occur locally, but verification can also be offloaded (completely or partially) onto a secure server. By offloading verification to a server beyond the reach of an attacker, the attacker cannot learn how to fabricate correct responses by studying the verification routine. In practice, the attestator and verifier can also be combined into one routine. Alternatively, their invocation can be pulled apart to some degree, in order to hide the dependency between them. However, there always has to remain a guarantee that whenever the attestator routine is invoked, so is the verifier routine, and vice versa.

3. **Update Tamper Detection Status**
   - **Details**: In step 3, which typically will follow immediately after step 2, the result of verification (the verdict) is used to encode the tamper detection status of the application. Based on the verdict, an update function is invoked that alters the delay data structures to encode that some form of tampering was or was not detected. It is important to note here that the update functions can also be invoked from random places in the original program, as long as those random invocations do not alter the information regarding detected tampering encoded in the data structures. This is important because such random updates will give the delay data structures the appearance of being integral data.

---

### Key Components

- **Attestators**:
  - Code guards
  - Timing
  - Data integrity
  - Control flow integrity

- **Verification**:
  - Local vs. remote
  - Prevent replay attacks

- **Delay Reaction**:
  - Attacker sees symptom
  - Hide relation with cause!

---

### Diagram

[Diagram showing the workflow and components of remote attestation]

---

### Notes

- **Remote attestation** refers to the process of verifying the integrity of applications remotely, often involving distributed systems and the use of attestors to ensure that the application's integrity is not compromised.

---

**Page 48 of 99**
Anti-disassembly

• Hide code
  • packers, virtualization, download code on demand, self-modifying code
• Junk bytes
• Indirect control flow transfers
• Jumps into middle of instructions
• Code layout randomization
• Overlapping instructions
• Exploit known heuristics
  • continuation points
  • patterns for function prologues, epilogues, calls, ...

Often, wrong information is worse than no information.
Anti-disassembly examples

Example 1

0x123a: jmp 0xabca;
...  
0xabca: addl #44,eax

Obfuscation

0x123a: call 0xabca;
...  
0xabca: pop ebx;
addl #44,eax

Example 2

0x123a: call 0xabca;
...  
0xabca: ... ret

Obfuscation

0x123a: push *(0xc000)
jmp 0xabca
pop eax
...

0xabca: ... jmp *(esp)

0xc000: 0x12424
Anti-decompilation

Exploit semantic gap between source code and assembly code or bytecode

• strip unnecessary symbol information
• rename identifiers (I,I,L,1)
• goto spaghetti
• disobey constructor conventions
• disobey exception handling conventions
Anti-decompilation example

```
pre();
try{
    might_throw_exception();
} catch(Exception e){
    handle_exception();
} post();
```
Anti-debugging

• Option 1: check environment for presence debugger

• Option 2: prevent debugger to attach
  • OS & hardware support at most one debugger per process
  • occupy one seat with custom “debugger” process
  • make control & data flow dependent on custom debugger

• anti-debugging by means of self-debugging
Self-Debugging

function 1

function 2

function 3

mini debugger
Self-Debugging

- function 1
- function 2
- function 3
- mini debugger
Self-Debugging

process 1045

function 1

function 2

function 3

mini debugger

debuggee

process 3721

function 1

function 2

function 3

mini debugger

degger
Self-Debugging

Process 1045

- Function 1
- Function 2a
- Function 3
- Mini debugger

Process 3721

- Function 1
- Function 2b
- Function 3
- Mini debugger

Debuggee

Debugger
Anti-emulation

• Emulators are buggy – incomplete
• Virtual environments are not real

• Johanna Rutkowska
  • Blue pill
  • Red pill
Lecture Overview

1. Basic Attacks
   • attacks on what?
   • basic attack tools & techniques

2. Defenses
   • anti Anything

3. Advanced Automated Attacks
   • generic deobfuscation
   • symbolic execution

4. Defenses
   • anti-even-more
Generic Deobfuscation (Yadegari et al IEEE S&P 2015)

• no obfuscation-specific assumptions
  • treat programs as input-to-output transformations
  • use semantics-preserving transformations to simplify execution traces
• dynamic analysis to handle runtime unpacking

Diagram:
- **Input program**
  - **Taint analysis (bit-level)**: map flow of values from input to output
  - **Semantics-preserving transformations / simplifications**: reconstruct logic of simplified computation
  - **Control flow reconstruction**: control flow graph
Generic Deobfuscation (Yadegari et al IEEE S&P 2015)

Instructions "tainted" as propagating values from input to output.

Input-to-output computation (further simplified)

Used to construct control flow graph.
Generic Deobfuscation (Yadegari et al IEEE S&P 2015)

- Quasi-invariant locations: locations that have the same value at each use.
- Their transformations:
  - Arithmetic simplification
    - adaptation of constant folding to execution traces
    - consider quasi-invariant locations as constants
    - controlled to avoid over-simplification
  - Control simplification
    - E.g., convert indirect jump through a quasi-invariant location into a direct jump
  - Data movement simplification
    - use pattern-driven rules to identify and simplify data movement.
  - Dead code elimination
    - need to consider implicit destinations, e.g., condition code flags.
Generic Deobfuscation (Yadegari et al IEEE S&P 2015)
Symbolic Execution

effective because most obfuscations implement semantics that do not involve input
Lecture Overview

1. Basic Attacks
   • attacks on what?
   • basic attack tools & techniques

2. Defenses
   • antianything

3. Advanced Automated Attacks
   • generic deobfuscation
   • symbolic execution

4. Defenses
   • anti-even-more
Anti-taint analysis

- tainting all data with artificial computations

- hiding data dependencies through covert channels
  - time
  - system state
  - anything not normally checked by analysis
Obfuscations with varying, input-dependent behavior (Banescu et al, ACSAC 2016)

Listing 3: Program with loop

```c
1   unsigned char *str = argv[1];
2   unsigned int hash = 0;
3   for(int i = 0; i < strlen(str); str++, i++) {
4       hash = (hash << 7) ^ (*str);
5   }
6   if (hash == 809267) printf("win\n");
```

1. RANGE Divider

Listing 4: Program from Listing 3 obfuscated with RANGE Divider

```c
1   unsigned char *str = argv[1];
2   unsigned int hash = 0;
3   for(int i = 0; i < strlen(str); str++, i++) {
4       char chr = *str;
5       if (chr > 42) {
6           hash = (hash << 7) ^ chr;
7       } else {
8           hash = (hash * 128) ^ chr;
9       }
10   }
11   if (hash == 809267) printf("win\n");
```
Obfuscations with varying, input-dependent behavior (Banescu et al, ACSAC 2016)

2. INPUT INVARIANTS

1. inject extra inputs into programs
2. let correct execution depend on invariant properties of those inputs

for example: feed program extra key to decrypt bytecode

(how to get these to the user ???)