ADVANCED ATTACK SURFACE DISCOVERY AND EXPLOITATION

Adrian Bednarek

Security Analyst/Researcher
Independent Security Evaluators
@ISEsecurity
Obligatory who is this guy?

- Adrian Bednarek
- Security Analyst/Researcher at ISE (Independent Security Evaluators)
- Started in the security field as an ethical blackhat (!?)
- Here to talk about emerging technologies in the battlefield of information security as it pertains to complex software used in many fields including IoT:
  - Custom protocols
  - Code obfuscation
  - Self modifying code
Defining the attack surface
External attack surface
Hidden attack surface
A simple application function code flow:
CODE AUDITS AND SYSTEM HARDENING
This should hold. I’ve looked over the code a dozen times.
Security Defects

- Various severities
- Configuration
- Code execution
- Business logic
- Authenticated users
- Unauthenticated users
Hate to spoil things, but...
The Arms Race

- Attack surfaces are increasing
  - Top layer (User and internet facing services)
  - Deep hidden layer (Business logic)

- Attack surfaces are layered
  - Top (‘script kiddies’, automated scanners)
  - Middle (Hobbyists, for profit individuals or groups)
  - Hidden layer (Highly skilled and motivated attackers using custom tools)
Typical Attack Flow

- Obtain product
- Protocol Analysis
  - Manual Network Vulnerability analysis
  - Fuzzing
  - Source/Binary Analysis
  - Weaponization (exploit development)
- Open source research
- Closed source research
An attackers arsenal

- Threat modeling
  - Inventory all the things that could be exploited

- Manual testing
  - Static code review
  - Network analysis

- Tool assisted testing
  - Dynamic code analysis
    - Debugging/Manual fuzzing
      - Automated fuzzing
  - Network MITM tools for dynamic analysis
Attack Surface Fuzzing

- **Manual Fuzzing**
  - Time consuming
  - Run tests with best guest inputs to trigger vulnerability discovery
    - Time consuming!
    - Especially when preexisting events must be established (e.g. complex state sessions)

- **Automated fuzzing**
  - Run many tests quickly and log abnormal results
Attack Surface Discovery

- Explore what?
  - Everything
  - Specific points of interest
    - Trigger events in hidden layer
Example Bug Hunting

```c
int readData(int fd)
{
    char header[50];
    char body[100];
    size_t size_header = 50;
    size_t size_body = 100;

    read(fd, header, size_body);
    read(fd, body, size_body);

    return 0;
}
```
Example Bug Hunting

```c
int readData(int fd)
{
    char header[50];
    char body[100];
    size_t size_header = 50;
    size_t size_body = 100;

    read(fd, header, size_body);
    read(fd, body, size_body);

    return 0;
}
```
Example Bug Hunting

```c
int readData(int fd)
{
    char header[50];
    char body[100];
    size_t size_header = 50;
    size_t size_body = 100;

    read(fd, header, 100); // Classic Buffer Overflow
    read(fd, body, 100);

    return 0;
}
```
Deep Dive

- Manual analysis of previous example
  - Would probably be missed
  - Time consuming to find
  - Counter intuitive

- Automated fuzzing
  - Discovered in seconds
    - All permutations will be tested
      - Leading to discovery of other classes of bugs!
Inputs

- Outwardly facing APIs
  - In comparison to the whole system, a small number
  - Frequently executed
  - Battle tested (hopefully!)
  - May have layers of obfuscation (hopefully)
    - Obfuscated solutions may be hard to audit
      - Low hanging fruit exploits may be out of reach
  - Trigger code deep within program logic
Finally!

Simple services are composed of
- Millions of lines of assembly
  - Composing thousands of functions
- Unrealistic to explore and fuzz everything
  - Especially when fuzzing requires stateful permutations

Automated discovery of code paths that touch data of interest
- Data that an attacker can input into the system
Custom tools to discover attack surfaces

- Custom tools and solutions are used by adversaries
  - In house solutions
  - Black market

- Thread Imager
  - Automatic discovery of code paths an adversary can influence
  - Attack surface discovery
  - Allows lower skilled adversaries to exploit complex and obfuscated systems
    - Encrypted code
    - Obfuscated code
    - Self modifying code
External APIs and inputs fire off numerous subsystems

Phelps & Mazziotta, UCLA
Summary

- Attack surfaces are multi faceted and multi layered
- Discovery of code paths that handle user input lead to an increase in attack surface
- Adversaries are capable of learning the inner workings of services at a very fine grained level – sometimes knowing more about the internal mechanics than the developers
Summary

- Deep attack surface discovery
  - Targeted attacks of specific discovered functionality
    - Less ‘noisy’
    - More likely to be exploited unnoticed
QUESTIONS?

About tools/Obfuscation effectiveness?/Anything?
THANK YOU!

Adrian Bednarek
Independent Security Evaluators
@ISEsecurity

https://www.linkedin.com/in/adrianbksd/