Secure Development After Security Bugs

Jeremy Epstein
Program Manager

Presentation to 1st
IEEE Cybersecurity Development Conference (SecDev)

11/03/16

Distribution Statement A. Approved for public release: distribution

Premise

Secure development is now more than just avoiding buffer overruns, SQL injections, etc.

It's about recognizing new classes of vulnerabilities

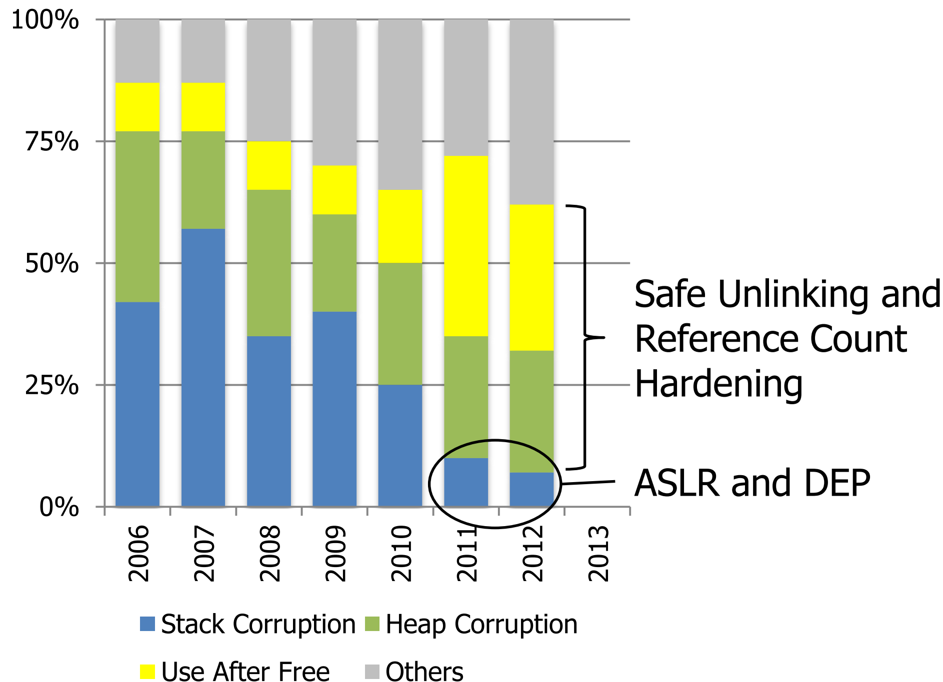
Secure software development has evolved over the last 20 years, but we're still looking for the same types of problems

- First generation (buffer overflow, sql injection, etc.) are well understood
- Second order issues (side channels, algorithmic exploits) are much more difficult to find and understand

Distribution Statement A. Approved for public release: distribution

The emerging class of algorithmic vulnerabilities

Past: memory vulnerabilities



Memory protections mitigate many implementation flaws, driving adversaries to exploit algorithmic vulnerabilities

Next: algorithmic vulnerabilities

1. Algorithmic complexity attacks

2011

Huge portions of the Web vulnerable to hashing denial-of-service attack
A flaw common to most popular Web programming languages can be used to launch …
by Jon Brodkin - Dec 28 2011, 2:25pm EST
ARS TECHNICA

High complexity loops and other algorithmic features having the potential to consume excessive resources and cause a denial of service

2. Side Channel Attacks

2013

Gone in 30 seconds: New attack plucks secrets from HTTPS-protected pages
Exploit called BREACH bypasses the SSL crypto scheme protecting millions of sites.
by Dan Goodin - Aug 1 2013, 11:30am EDT
ARS TECHNICA

Externally observable behavioral features that have the potential to leak internal state information

Chart: prevalence of classes of vulnerabilities Microsoft knows have been exploited in their software.
SOURCE: Microsoft 2013 Software Vulnerability Exploitation Trends report.

Distribution Statement A. Approved for public release: distribution

Canonical example: Bad password-check side channel

An adversary can deduce information about the password from the amount of time it takes to reject incorrect guesses

Actual password:	S	E	C	R	E	T
Adversary's guess:	S	E	C	O	N	D

Adversary guesses one character at a time and observes response

Bad implementation: ✓ ✓ ✓ ✗

rejecting on ✗ creates a timing side channel

Good implementation: ✓ ✓ ✓ ✗ ✗ ✗

reject after checking all input – no side channel

STAC tools need to distinguish between these cases

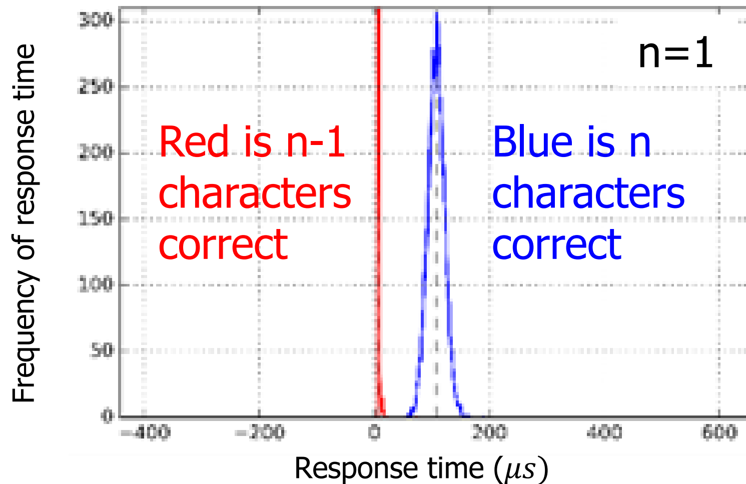
In 2013 researchers discovered a way to use timing to rapidly guess passwords for the Bitcoin digital currency software

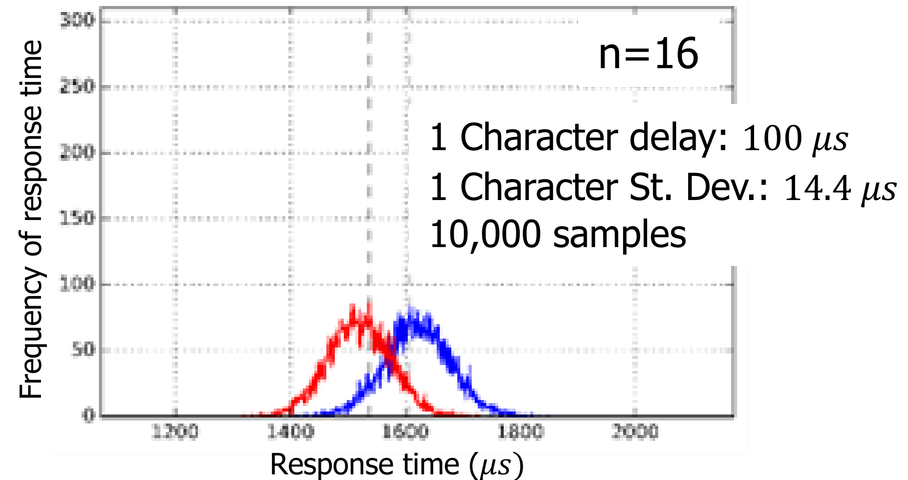
CVE 2013-4165 in the National Vulnerability Database

Source: Apogee Research, used with permission

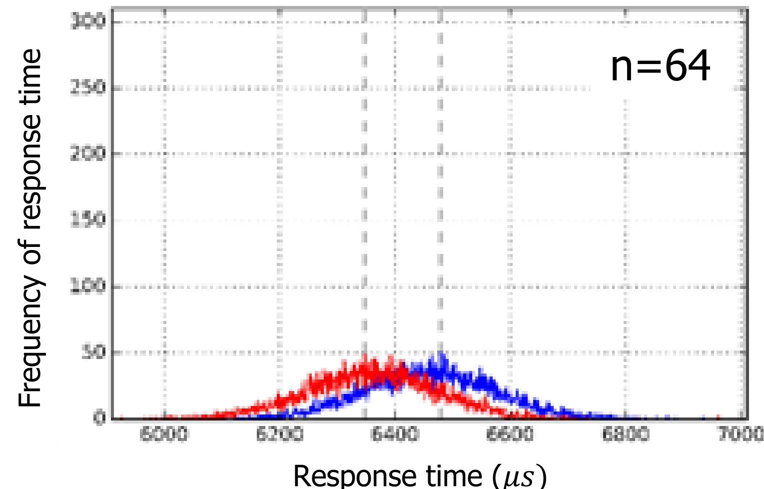
Resource is a vital concept in algorithmic exploitation

Side channels (like password timing) may not be practically exploitable
Defining an attacker input budget makes algorithmic vulnerability questions concrete





- 1 character: easily exploitable
- 16 characters: exploitable in moderate time with high probability
- 64 characters: exploit difficult because noise overwhelms signal



Source: Apogee Research, used with permission

Canonical vulnerable example and questions

Sample program

```
for(int i=0; i<n; i++)
    consume 1000 ms
    for(int t=0; t<n; t++)
        consume 1 ms
```

Sample question

Input budget: n<= 99
Resource consumption: time < 60s

Q1: Are there a set of inputs (n<=99) that cause it to exceed resource limits (60s)?

Q2: Are there inputs (n<=99) that cause the value of n to be leaked as a space sidechannel?

Q1: If only look at highest complexity loop: consumes 99*99*1ms = 9.8s => NO

Complete analysis consumes 99*1000ms + 99*99*1ms = 109s => YES

Must consider overall resource consumption

Q2: No, because nothing shows space usage (e.g., disk, memory)

Source: Apogee Research, used with permission

Only best and worst case considered

Some analysis looking for only best/worst case times missed differential resource consumption that may contain a side channel.

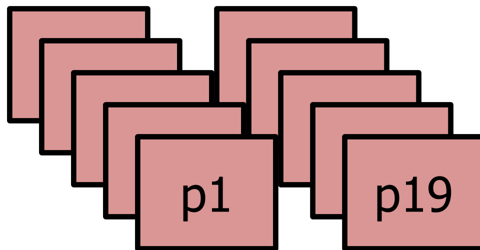
In code below, if T==2, then there is a time differential if guess <=secret vs guess>secret

```
if(guess <= Secret) {
    if (T == 1) {Thread.sleep(1);}                                        // Const
    else if (T == 2){ for(int i = 0; i < n; i++){Thread.sleep(1);}        // O(n)
    else { for(int i = 0; i < n*n*n; i++){Thread.sleep(1);}               // O(n³)
} else {
    if (T == 1) {Thread.sleep(1);}                                        // Const
    else if (T == 2){ for(int i = 0; i < n*n; i++){Thread.sleep(1);}      // O(n²)
    else { for(int i = 0; i < n*n*n; i++){Thread.sleep(1);}               // O(n³)
}
```

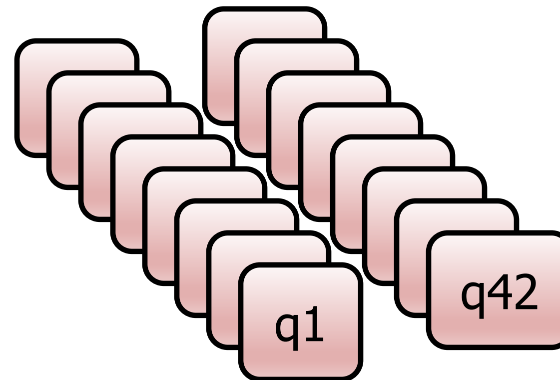
Source: Apogee Research, used with permission

Competitive engagement overview

Two red teams create programs



and a set of questions about vulnerability



At engagement, these are given to
8 research teams
1 control team

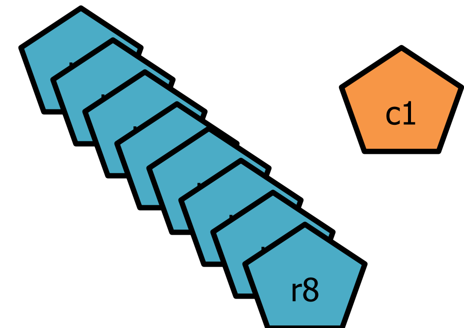
Results scored by engagement lead

Yes Answers	No Answers	Yes Accuracy	No Accuracy
20	22	90%	77%
23	6	74%	83%
18	24	83%	67%
12	2	92%	50%
5	5	100%	60%

Yes = vulnerable + justification

Who then analyze and answer

No = not vulnerable + justification



Distribution Statement A. Approved for public release: distribution

Gabfeed: A sample program from STAC (researcher's perspective)

- Description: web forum software that allows community members to post messages, search posted messages, and provides additional information on selected ('special') topics

Source: CyberPoint International, used with permission

True vulnerability #1: Timing side channel

- Description
  - Allows a third party to discover the private key used by the server to authenticate itself to a user. Users are provided with a script which accepts a numerical input (u) and calculates an input to be sent to the server (U = g^u mod m) and the expected server response (S = g^us mod m). Users authenticate the server by navigating to https://localhost:8080/authenticate and submitting the value "U". Server calculates the value of "U^s mod m" and returns it to the user. User confirms that the server's output and the expected server response are the same. Vulnerability lies in the server's calculation of "U^s mod m". The server iterates through the bits of the server secret key and performs an expensive operation when the bit is 1 vs when the bit is 0.
- Challenge Question: Is there a side channel in time in the challenge program from which a third party can discover the server's private key used in the server's authentication?
- Available Operations:
  - Passive Operations: observe the encrypted packets within a single user session or authentication of the server (cost 1 per session)
  - Active Operations: Issue an HTTP request and receive a response. (cost 1 per operation)
- Operational Budget: Max number of operations : 60000; Probability of success: 95%

Source: CyberPoint International, used with permission

True vulnerability #2: algorithmic complexity (time)

- Description
  - Vulnerability is in the shortest path line breaking algorithm; has O (n*width) behavior, where n is the number of words and width is the width of the client browser in pixels – used to inform the server of when to line-break the text.
  - Normally, there is a JavaScript function that finds the width (in pixels) of the column where the text will be placed. The function issues a ‘PUT’ command to https://localhost:8080/width<width>, which sets the width. Using this method the width should never be large enough to cause the resource usage limit to be exceeded. An attacker can call the ‘PUT’ with a large width setting and post a corresponding long message. This will cause the resource usage limit to be exceeded.
- Challenge Question: Is there an algorithmic complexity vulnerability in time that would cause the challenge program's real runtime to exceed the resource usage limit given the input budget?
- Input Budget: Maximum sum of the PDU sizes of the HTTP requests sent from the user to the server: 400000 bytes
- Resource Usage Limit: Runtime: 300 seconds measured as the time interval between sending the last byte of the attack and the server sending the last byte of the response.

Source: CyberPoint International, used with permission

False vulnerability #1

- Challenge question: Is there an algorithmic complexity in space that would cause the challenge program's memory usage to exceed the resource usage limit in one user session given the input budget?
- Note: Although the program is a web based application that supports multiple users, this question assumes there is only one user interacting with the system in any vulnerability demonstration.
- Input Budget:
  - Maximum sum of the PDU sizes of the HTTP requests sent from the user to the server: 400000 bytes
- Resource Usage Limit:
  - Available Memory: 512 MB (total memory usage of the JVM shown under the RSS column in pmap -x <PID>)

Source: CyberPoint International, used with permission

Distribution Statement A. Approved for public release: distribution

How do you measure bad enough?

Input Budget

Driver 1: Order

Resource Usage Limit

Driver 2: Coefficients

Driver 3: Data Guards

Vulnerable Performance

Non-Vulnerable Performance

Intended

Run-Time (Time or Space)

4500
4000
3500
3000
2500
2000
1500
1000
500
0

0 100 200 300 400 500

Input Size

Source: Apogee Research, used with permission

Distribution Statement A. Approved for public release: distribution

Study results (iteration 3)

Source: Apogee Research, used with permission

STAC program analysis approaches

Existing techniques (used by most teams)

- static/dynamic analysis
- information flow analysis
- taint analysis

Researcher techniques

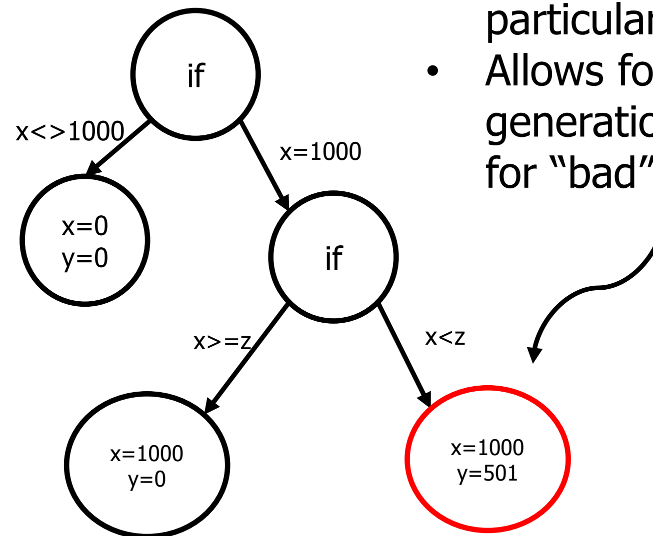
- symbolic execution
- bytecode cost modeling
- program reduction
- loop summarization
- complexity bounding
- control flow graphs

Example of Symbolic Execution

```
int z = 2*y;
if (x == 1000) {
  if (x < z) {
    fail();
  }
}
```

How to reach fail()?

- Represent program as graph
- Use symbols for branches
- Use constraint solver to find inputs forcing particular paths
- Allows for automatic generation of test cases for “bad” behavior



Challenges

- Getting research teams to answer “not vulnerable”
- Balancing research and engagements
- Keeping up with technology evolution
- Defining boundaries for challenge problems
- Modelling complexity
  - large 3rd party libraries
  - JNI (native code)
  - Garbage collection

Case study: Are side channels important?

- Imagine a website where the length of the response indicates product ordered
  - 20 bytes/μs = cellphone
  - 22 bytes/μs = refrigerator
  - 24 bytes/μs = breakfast cereal
  - *Sensitivity = low*

- Same software, used for over the counter pharmacy products
  - 20 bytes/μs = antacids
  - 22 bytes/μs = paternity test
  - 24 bytes/μs = AIDS test
  - *Sensitivity = medium*

- Same software, used for prescription medicines
  - 20 bytes/μs = Melanoma treatments
  - 22 bytes/μs = “Plan B”
  - 24 bytes/μs = Aciclovir
  - *Sensitivity = high*

Distribution Statement A. Approved for public release: distribution

But what should we do?

- Constant time for all operations?
- Constant space for all operations?
- Introduce increased noise?

Distribution Statement A. Approved for public release: distribution

How do we actually build more secure programs?

Enhancing tools and training for developers / analysts

AC/SC issues can emerge from design layers (OS/JVM/3rd party lib/code). Need analysis that can span technologies (bytecode -> binary, binary -> hardware)

Learn when mitigations such as constant time algorithms are useful

Architectural analysis tools that can model timing issues?

Distribution Statement A. Approved for public release: distribution

www.darpa.mil