CWE-489: Active Debug Code

Weakness ID: 489
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

The product is deployed to unauthorized actors with debugging code still enabled or active, which can create unintended entry points or expose sensitive information.

Extended Description

A common development practice is to add "back door" code specifically designed for debugging or testing purposes that is not intended to be shipped or deployed with the product. These back door entry points create security risks because they are not considered during design or testing and fall outside of the expected operating conditions of the product.

Alternate Terms

Leftover debug code: This term originates from Seven Pernicious Kingdoms

Common Consequences

Scope	Impact	Likelihood
Confidentiality Integrity Availability Access Control Other	**Technical Impact:** *Bypass Protection Mechanism; Read Application Data; Gain Privileges or Assume Identity; Varies by Context* The severity of the exposed debug application will depend on the particular instance. At the least, it will give an attacker sensitive information about the settings and mechanics of web applications on the server. At worst, as is often the case, the debug application will allow an attacker complete control over the web application and server, as well as confidential information that either of these access.	

Potential Mitigations

Phases: Build and Compilation; Distribution

Remove debug code before deploying the application.

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	P	710	Improper Adherence to Coding Standards
ParentOf	V	11	ASP.NET Misconfiguration: Creating Debug Binary
CanPrecede	B	215	Insertion of Sensitive Information Into Debugging Code

Relevant to the view "Software Development" (CWE-699)

Nature	Type	ID	Name
MemberOf	C	1006	Bad Coding Practices

Modes Of Introduction

Phase	Note
Implementation	In web-based applications, debug code is used to test and modify web application properties, configuration information, and functions. If a debug application is left on a production server, this oversight during the "software process" allows attackers access to debug functionality.
Build and Compilation	
Operation	

Applicable Platforms

Languages

Class: Not Language-Specific *(Undetermined Prevalence)*

Technologies

Class: Not Technology-Specific *(Undetermined Prevalence)*
Class: ICS/OT *(Undetermined Prevalence)*

Demonstrative Examples

Example 1

Debug code can be used to bypass authentication. For example, suppose an application has a login script that receives a username and a password. Assume also that a third, optional, parameter, called "debug", is interpreted by the script as requesting a switch to debug mode, and that when this parameter is given the username and password are not checked. In such a case, it is very simple to bypass the authentication process if the special behavior of the application regarding the debug parameter is known. In a case where the form is:

Example Language: **HTML** *(bad code)*

```
<FORM ACTION="/authenticate_login.cgi">
  <INPUT TYPE=TEXT name=username>
  <INPUT TYPE=PASSWORD name=password>
  <INPUT TYPE=SUBMIT>
</FORM>
```

Then a conforming link will look like:

(informative)

```
http://TARGET/authenticate_login.cgi?username=...&password=...
```

An attacker can change this to:

(attack code)

```
http://TARGET/authenticate_login.cgi?username=&password=&debug=1
```

Which will grant the attacker access to the site, bypassing the authentication process.

Weakness Ordinalities

Ordinality	Description
Indirect	*(where the weakness is a quality issue that might indirectly make it easier to introduce security-relevant weaknesses or make them more difficult to detect)*
Primary	*(where the weakness exists independent of other weaknesses)*

Detection Methods

Automated Static Analysis

Automated static analysis, commonly referred to as Static Application Security Testing (SAST), can find some instances of this weakness by analyzing source code (or binary/compiled code) without having to execute it. Typically, this is done by building a model of data flow and control flow, then searching for potentially-vulnerable patterns that connect "sources" (origins of input) with "sinks" (destinations where the data interacts with external components, a lower layer such as the OS, etc.)

Effectiveness: High

Memberships

Nature	Type	ID	Name
MemberOf	C	485	7PK - Encapsulation
MemberOf	C	731	OWASP Top Ten 2004 Category A10 - Insecure Configuration Management
MemberOf	C	1002	SFP Secondary Cluster: Unexpected Entry Points
MemberOf	C	1371	ICS Supply Chain: Poorly Documented or Undocumented Features
MemberOf	C	1412	Comprehensive Categorization: Poor Coding Practices

Vulnerability Mapping Notes

Usage: ALLOWED *(this CWE ID could be used to map to real-world vulnerabilities)*

Reason: Acceptable-Use

Rationale:

This CWE entry is at the Base level of abstraction, which is a preferred level of abstraction for mapping to the root causes of vulnerabilities.

Comments:

Carefully read both the name and description to ensure that this mapping is an appropriate fit. Do not try to 'force' a mapping to a lower-level Base/Variant simply to comply with this preferred level of abstraction.

Notes

Other

In J2EE a main method may be a good indicator that debug code has been left in the application, although there may not be any direct security impact.

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Leftover Debug Code
OWASP Top Ten 2004	A10	CWE More Specific	Insecure Configuration Management
Software Fault Patterns	SFP28		Unexpected access points

Related Attack Patterns

CAPEC-ID	Attack Pattern Name
CAPEC-121	Exploit Non-Production Interfaces
CAPEC-661	Root/Jailbreak Detection Evasion via Debugging

References

[REF-6] Katrina Tsipenyuk, Brian Chess and Gary McGraw. "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors". NIST Workshop on Software Security Assurance Tools Techniques and Metrics. NIST. 2005-11-07. <https://samate.nist.gov/SSATTM_Content/papers/Seven%20Pernicious%20Kingdoms%20-%20Taxonomy%20of%20Sw%20Security%20Errors%20-%20Tsipenyuk%20-%20Chess%20-%20McGraw.pdf>.

Content History

Submissions

Submission Date	Submitter	Organization
2006-07-19 *(CWE Draft 3, 2006-07-19)*	7 Pernicious Kingdoms	

Modifications

Previous Entry Names

Page Last Updated: July 16, 2024