android_bootable_recovery Scan Report

Project Name	android_bootable_recovery
Scan Start	Friday, June 21, 2024 11:43:23 PM
Preset	Checkmarx Default
Scan Time	00h:17m:02s
Lines Of Code Scanned	13249
Files Scanned	15
Report Creation Time	Saturday, June 22, 2024 12:05:15 AM
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073
Team	CxServer
Checkmarx Version	8.7.0
Scan Type	Full
Source Origin	LocalPath
Density	1/100 (Vulnerabilities/LOC)
Visibility	Public

Filter Settings

Severity

Included: High, Medium, Low, Information

Excluded: None

Result State

Included: Confirmed, Not Exploitable, To Verify, Urgent, Proposed Not Exploitable

Excluded: None

Assigned to

Included: All

Categories

Included:

Uncategorized	All
Custom	All
PCI DSS v3.2	All
OWASP Top 10 2013	All
FISMA 2014	All
NIST SP 800-53	All
OWASP Top 10 2017	All
OWASP Mobile Top 10 2016	All

Excluded:

Uncategorized	None
Custom	None
PCI DSS v3.2	None
OWASP Top 10 2013	None
FISMA 2014	None

NIST SP 800-53	None
OWASP Top 10 2017	None
OWASP Mobile Top 10 2016	None

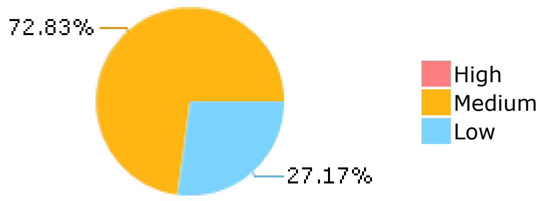
Results Limit

Results limit per query was set to 50

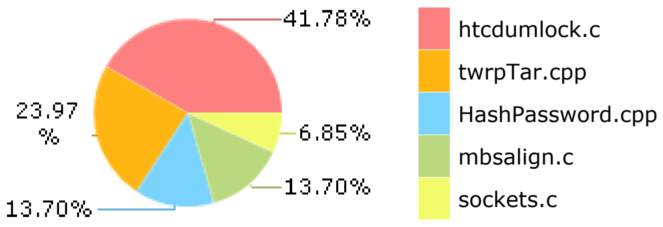
Selected Queries

Selected queries are listed in Result Summary

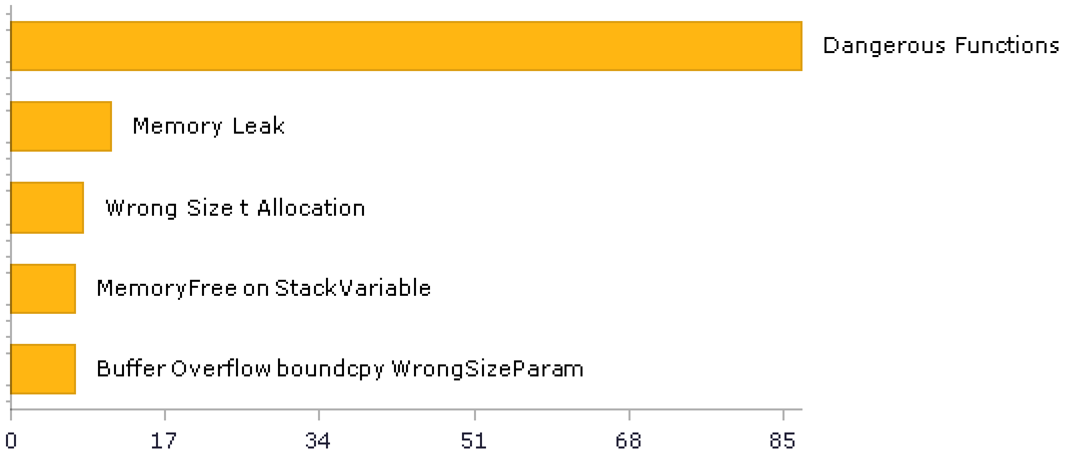
Result Summary



Most Vulnerable Files



Top 5 Vulnerabilities



Scan Summary - OWASP Top 10 2017

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2017

Category	Threat Agent	Exploitability	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection	App. Specific	EASY	COMMON	EASY	SEVERE	App. Specific	15	11
A2-Broken Authentication	App. Specific	EASY	COMMON	AVERAGE	SEVERE	App. Specific	15	15
A3-Sensitive Data Exposure	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	App. Specific	2	2
A4-XML External Entities (XXE)	App. Specific	AVERAGE	COMMON	EASY	SEVERE	App. Specific	0	0
A5-Broken Access Control*	App. Specific	AVERAGE	COMMON	AVERAGE	SEVERE	App. Specific	0	0
A6-Security Misconfiguration	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	0	0
A7-Cross-Site Scripting (XSS)	App. Specific	EASY	WIDESPREAD	EASY	MODERATE	App. Specific	0	0
A8-Insecure Deserialization	App. Specific	DIFFICULT	COMMON	AVERAGE	SEVERE	App. Specific	0	0
A9-Using Components with Known Vulnerabilities*	App. Specific	AVERAGE	WIDESPREAD	AVERAGE	MODERATE	App. Specific	87	87
A10-Insufficient Logging & Monitoring	App. Specific	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	App. Specific	0	0

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Top 10 2013

Further details and elaboration about vulnerabilities and risks can be found at: OWASP Top 10 2013

Category	Threat Agent	Attack Vectors	Weakness Prevalence	Weakness Detectability	Technical Impact	Business Impact	Issues Found	Best Fix Locations
A1-Injection	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	AVERAGE	SEVERE	ALL DATA	0	0
A2-Broken Authentication and Session Management	EXTERNAL, INTERNAL USERS	AVERAGE	WIDESPREAD	AVERAGE	SEVERE	AFFECTED DATA AND FUNCTIONS	0	0
A3-Cross-Site Scripting (XSS)	EXTERNAL, INTERNAL, ADMIN USERS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE	AFFECTED DATA AND SYSTEM	0	0
A4-Insecure Direct Object References	SYSTEM USERS	EASY	COMMON	EASY	MODERATE	EXPOSED DATA	0	0
A5-Security Misconfiguration	EXTERNAL, INTERNAL, ADMIN USERS	EASY	COMMON	EASY	MODERATE	ALL DATA AND SYSTEM	0	0
A6-Sensitive Data Exposure	EXTERNAL, INTERNAL, ADMIN USERS, USERS BROWSERS	DIFFICULT	UNCOMMON	AVERAGE	SEVERE	EXPOSED DATA	1	1
A7-Missing Function Level Access Control*	EXTERNAL, INTERNAL USERS	EASY	COMMON	AVERAGE	MODERATE	EXPOSED DATA AND FUNCTIONS	0	0
A8-Cross-Site Request Forgery (CSRF)	USERS BROWSERS	AVERAGE	COMMON	EASY	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0
A9-Using Components with Known Vulnerabilities*	EXTERNAL USERS, AUTOMATED TOOLS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	87	87
A10-Unvalidated Redirects and Forwards	USERS BROWSERS	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE	AFFECTED DATA AND FUNCTIONS	0	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - PCI DSS v3.2

Category	Issues Found	Best Fix Locations
PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection	1	1
PCI DSS (3.2) - 6.5.2 - Buffer overflows	10	10
PCI DSS (3.2) - 6.5.3 - Insecure cryptographic storage	0	0
PCI DSS (3.2) - 6.5.4 - Insecure communications	0	0
PCI DSS (3.2) - 6.5.5 - Improper error handling*****	0	0
PCI DSS (3.2) - 6.5.7 - Cross-site scripting (XSS)	0	0
PCI DSS (3.2) - 6.5.8 - Improper access control	0	0
PCI DSS (3.2) - 6.5.9 - Cross-site request forgery	0	0
PCI DSS (3.2) - 6.5.10 - Broken authentication and session management	0	0

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - FISMA 2014

Category	Description	Issues Found	Best Fix Locations
Access Control	Organizations must limit information system access to authorized users, processes acting on behalf of authorized users, or devices (including other information systems) and to the types of transactions and functions that authorized users are permitted to exercise.	8	8
Audit And Accountability*	Organizations must: (i) create, protect, and retain information system audit records to the extent needed to enable the monitoring, analysis, investigation, and reporting of unlawful, unauthorized, or inappropriate information system activity; and (ii) ensure that the actions of individual information system users can be uniquely traced to those users so they can be held accountable for their actions.	0	0
Configuration Management	Organizations must: (i) establish and maintain baseline configurations and inventories of organizational information systems (including hardware, software, firmware, and documentation) throughout the respective system development life cycles; and (ii) establish and enforce security configuration settings for information technology products employed in organizational information systems.	0	0
Identification And Authentication*	Organizations must identify information system users, processes acting on behalf of users, or devices and authenticate (or verify) the identities of those users, processes, or devices, as a prerequisite to allowing access to organizational information systems.	7	7
Media Protection	Organizations must: (i) protect information system media, both paper and digital; (ii) limit access to information on information system media to authorized users; and (iii) sanitize or destroy information system media before disposal or release for reuse.	1	1
System And Communications Protection	Organizations must: (i) monitor, control, and protect organizational communications (i.e., information transmitted or received by organizational information systems) at the external boundaries and key internal boundaries of the information systems; and (ii) employ architectural designs, software development techniques, and systems engineering principles that promote effective information security within organizational information systems.	0	0
System And Information Integrity	Organizations must: (i) identify, report, and correct information and information system flaws in a timely manner; (ii) provide protection from malicious code at appropriate locations within organizational information systems; and (iii) monitor information system security alerts and advisories and take appropriate actions in response.	3	3

* Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - NIST SP 800-53

Category	Issues Found	Best Fix Locations
AC-12 Session Termination (P2)	0	0
AC-3 Access Enforcement (P1)	15	15
AC-4 Information Flow Enforcement (P1)	0	0
AC-6 Least Privilege (P1)	0	0
AU-9 Protection of Audit Information (P1)	0	0
CM-6 Configuration Settings (P2)	0	0
IA-5 Authenticator Management (P1)	0	0
IA-6 Authenticator Feedback (P2)	0	0
IA-8 Identification and Authentication (Non-Organizational Users) (P1)	0	0
SC-12 Cryptographic Key Establishment and Management (P1)	0	0
SC-13 Cryptographic Protection (P1)	0	0
SC-17 Public Key Infrastructure Certificates (P1)	0	0
SC-18 Mobile Code (P2)	0	0
SC-23 Session Authenticity (P1)*****	0	0
SC-28 Protection of Information at Rest (P1)	0	0
SC-4 Information in Shared Resources (P1)	2	2
SC-5 Denial of Service Protection (P1)*****	16	16
SC-8 Transmission Confidentiality and Integrity (P1)	0	0
SI-10 Information Input Validation (P1)*****	9	5
SI-11 Error Handling (P2)*****	20	20
SI-15 Information Output Filtering (P0)	0	0
SI-16 Memory Protection (P1)	1	1

***** Project scan results do not include all relevant queries. Presets and\or Filters should be changed to include all relevant standard queries.

Scan Summary - OWASP Mobile Top 10 2016

Category	Description	Issues Found	Best Fix Locations
M1-Improper Platform Usage	This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.	0	0
M2-Insecure Data Storage	This category covers insecure data storage and unintended data leakage.	0	0
M3-Insecure Communication	This category covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.	0	0
M4-Insecure Authentication	This category captures notions of authenticating the end user or bad session management. This can include: -Failing to identify the user at all when that should be required -Failure to maintain the user's identity when it is required -Weaknesses in session management	0	0
M5-Insufficient Cryptography	The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2. This category is for issues where cryptography was attempted, but it wasnt done correctly.	0	0
M6-Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.	0	0
M7-Client Code Quality	This category is the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.	0	0
M8-Code Tampering	This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or	0	0

	modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.		
M9-Reverse Engineering	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.	0	0
M10-Extraneous Functionality	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.	0	0

Scan Summary - Custom

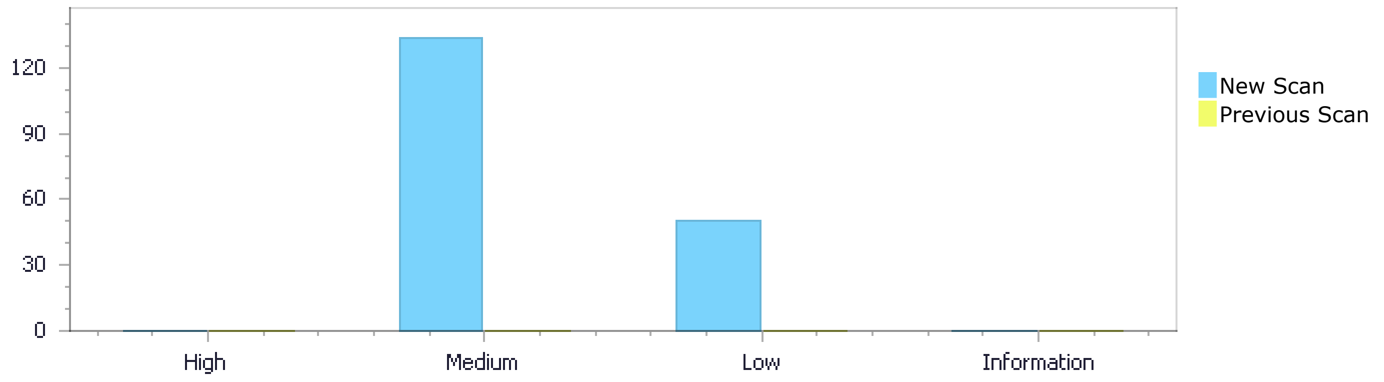
Category	Issues Found	Best Fix Locations
Must audit	0	0
Check	0	0
Optional	0	0

Results Distribution By Status

First scan of the project

	High	Medium	Low	Information	Total
New Issues	0	134	50	0	184
Recurrent Issues	0	0	0	0	0
Total	0	134	50	0	184

Fixed Issues	0	0	0	0	0



Results Distribution By State

	High	Medium	Low	Information	Total
Confirmed	0	0	0	0	0
Not Exploitable	0	0	0	0	0
To Verify	0	134	50	0	184
Urgent	0	0	0	0	0
Proposed Not Exploitable	0	0	0	0	0
Total	0	134	50	0	184

Result Summary

Vulnerability Type	Occurrences	Severity
Dangerous Functions	87	Medium
Memory Leak	11	Medium
Wrong Size t Allocation	8	Medium
Buffer Overflow boundcpy WrongSizeParam	7	Medium
MemoryFree on StackVariable	7	Medium

Stored Buffer Overflow cpycat	5	Medium
Long Overflow	2	Medium
Use of Uninitialized Variable	2	Medium
Divide By Zero	1	Medium
Heap Inspection	1	Medium
Integer Overflow	1	Medium
Stored Buffer Overflow boundcpy	1	Medium
Use of Zero Initialized Pointer	1	Medium
Unchecked Return Value	20	Low
TOCTOU	9	Low
Incorrect Permission Assignment For Critical Resources	8	Low
Improper Resource Access Authorization	7	Low
Inconsistent Implementations	1	Low
Insecure Temporary File	1	Low
NULL Pointer Dereference	1	Low
Potential Off by One Error in Loops	1	Low
Unreleased Resource Leak	1	Low
Use of Sizeof On a Pointer Type	1	Low

10 Most Vulnerable Files

High and Medium Vulnerabilities

File Name	Issues Found
android_bootable_recovery/htcdumlock.c	46
android_bootable_recovery/HashPassword.cpp	18
android_bootable_recovery/twrpTar.cpp	17
android_bootable_recovery/mbsalign.c	17
android_bootable_recovery/sockets.c	9
android_bootable_recovery/snprintf.c	7
android_bootable_recovery/main.c	6
android_bootable_recovery/ask.c	5
android_bootable_recovery/glob.c	5
android_bootable_recovery/fileutils.c	3

Scan Results Details

Dangerous Functions

Query Path:
CPP\Cx\CPP Medium Threat\Dangerous Functions Version:1

Categories

OWASP Top 10 2013: A9-Using Components with Known Vulnerabilities
OWASP Top 10 2017: A9-Using Components with Known Vulnerabilities

Description

Dangerous Functions\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=53
Status	New

The dangerous function, memcpy, was found in use at line 38 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	42	42
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
42.    memcpy((void*)buffer, (void*)prefix, strlen(prefix));
```

Dangerous Functions\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=54
Status	New

The dangerous function, memcpy, was found in use at line 38 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	44	44
Object	memcpy	memcpy

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
44.    memcpy((void*)ptr, key, key_size);
```

Dangerous Functions\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=55
Status	New

The dangerous function, memcpy, was found in use at line 38 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	53	53
Object	memcpy	memcpy

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
53.    memcpy(ret, (void*)&hash[0], SHA512_DIGEST_LENGTH);
```

Dangerous Functions\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=56
Status	New

The dangerous function, memcpy, was found in use at line 57 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	61	61
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
61.    memcpy((void*)buffer, (void*)prefix, strlen(prefix));
```

Dangerous Functions\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=57
Status	New

The dangerous function, memcpy, was found in use at line 57 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	63	63
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
63.    memcpy((void*)ptr, key, key_size);
```

Dangerous Functions\Path 6:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=500

	73&pathid=58
Status	New

The dangerous function, memcpy, was found in use at line 104 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	147	147
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
147.        memcpy(ramdisk, p, sizeof(char) * SCAN_SIZE);
```

Dangerous Functions\Path 7:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=59
Status	New

The dangerous function, memcpy, was found in use at line 169 in android_bootable_recovery/iso9660.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/iso9660.c	android_bootable_recovery/iso9660.c
Line	183	183
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/iso9660.c
Method int probe_iso9660(blkid_probe pr, const struct blkid_idmag *mag)

```
....
183.        memcpy(label, iso->volume_id, sizeof(label));
```

Dangerous Functions\Path 8:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=60
Status	New

The dangerous function, memcpy, was found in use at line 342 in android_bootable_recovery/mbsalign.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	456	456
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method mbsalign (const char *src, char *dest, size_t dest_size,

```
....
456.         dest = memcpy (dest, str_to_print, min (n_used_bytes, 
space_left));
```

Dangerous Functions\Path 9:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=61
Status	New

The dangerous function, memcpy, was found in use at line 114 in android_bootable_recovery/mbsalign.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	166	166
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
166.                         memcpy(r, p, len);
```

Dangerous Functions\Path 10:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=62
Status	New

The dangerous function, memcpy, was found in use at line 597 in android_bootable_recovery/sockets.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c
Line	613	613
Object	memcpy	memcpy

Code Snippet
File Name android_bootable_recovery/sockets.c
Method static int smart_socket_enqueue(asocket *s, apacket *p)

```
....
613.          memcpy(s->pkt_first->data + s->pkt_first->len,
```

Dangerous Functions\Path 11:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=63
Status	New

The dangerous function, sprintf, was found in use at line 57 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	72	72
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
72.          sprintf(hex_hash + (index * 2), "%02X", hash[index]);
```

Dangerous Functions\Path 12:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=64
Status	New

The dangerous function, sprintf, was found in use at line 83 in android_bootable_recovery/HashPassword.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	108	108
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	std::string PersonalizedHashSP800(const char* label, const char* context, const char* key, const size_t key_size) {

```
....
108.                sprintf(hex_hash + (index * 2), "%02x",
output[index]);
```

Dangerous Functions\Path 13:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=65
Status	New

The dangerous function, sprintf, was found in use at line 114 in android_bootable_recovery/mbsalign.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	131	131
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/mbsalign.c
Method	char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
131.                        sprintf(r, "\\x%02x", (unsigned char) *p);
```

Dangerous Functions\Path 14:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=66
Status	New

The dangerous function, sprintf, was found in use at line 114 in android_bootable_recovery/mbsalign.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	151	151
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
151.                                sprintf(r, "\\x%02x", (unsigned
char) *p);
```

Dangerous Functions\Path 15:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=67
Status	New

The dangerous function, sprintf, was found in use at line 114 in android_bootable_recovery/mbsalign.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	161	161
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
161.                                sprintf(r, "\\x%02x", (unsigned
char) *p);
```

Dangerous Functions\Path 16:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=68
Status	New

The dangerous function, sprintf, was found in use at line 691 in android_bootable_recovery/snprintf.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination

File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	734	734
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/snprintf.c
Method int main (void)

```
....
734.          sprintf (buf2, fp_fmt[x], fp_nums[y]);
```

Dangerous Functions\Path 17:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=69
Status	New

The dangerous function, sprintf, was found in use at line 691 in android_bootable_recovery/snprintf.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	748	748
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/snprintf.c
Method int main (void)

```
....
748.          sprintf (buf2, int_fmt[x], int_nums[y]);
```

Dangerous Functions\Path 18:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=70
Status	New

The dangerous function, sprintf, was found in use at line 482 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	567	567

Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::extractTarFork() {

```
....
567.                                    sprintf(actual_filename,
temp.c_str(), i, 0);
```

Dangerous Functions\Path 19:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=71
Status	New

The dangerous function, sprintf, was found in use at line 763 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	774	774
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::tarList(std::vector<TarListStruct> *TarList, unsigned thread_id) {

```
....
774.            sprintf(actual_filename, temp.c_str(), thread_id,
archive_count);
```

Dangerous Functions\Path 20:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=72
Status	New

The dangerous function, sprintf, was found in use at line 763 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	812	812
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::tarList(std::vector<TarListStruct> *TarList, unsigned thread_id) {

```
....
812.                              sprintf(actual_filename,
temp.c_str(), thread_id, archive_count);
```

Dangerous Functions\Path 21:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=73
Status	New

The dangerous function, sprintf, was found in use at line 853 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	858	858
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method void* twrpTar::extractMulti(void *cookie) {

```
....
858.          sprintf(actual_filename, temp.c_str(), threadTar->thread_id,
archive_count);
```

Dangerous Functions\Path 22:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=74
Status	New

The dangerous function, sprintf, was found in use at line 853 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	868	868
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method void* twrpTar::extractMulti(void *cookie) {

```
....
868.                 sprintf(actual_filename, temp.c_str(), threadTar-
>thread_id, archive_count);
```

Dangerous Functions\Path 23:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=75
Status	New

The dangerous function, sprintf, was found in use at line 1466 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1481	1481
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method unsigned long long twrpTar::get_size() {

```
....
1481.                sprintf(actual_filename, temp.c_str(), thread_id,
archive_count);
```

Dangerous Functions\Path 24:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=76
Status	New

The dangerous function, sprintf, was found in use at line 1466 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1489	1489
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	unsigned long long twrpTar::get_size() {

```
....
1489.                    sprintf(actual_filename, temp.c_str(), i,
archive_count);
```

Dangerous Functions\Path 25:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=77
Status	New

The dangerous function, sprintf, was found in use at line 1466 in android_bootable_recovery/twrpTar.cpp file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1495	1495
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	unsigned long long twrpTar::get_size() {

```
....
1495.                          sprintf(actual_filename,
temp.c_str(), i, archive_count);
```

Dangerous Functions\Path 26:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=78
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	184	184
Object	strcat	strcat

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c

Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
184.         strcat(twrp_device_path, device_id);
```

Dangerous Functions\Path 27:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=79
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	190	190
Object	strcat	strcat

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
190.         strcat(recovery_path, "/recovery");
```

Dangerous Functions\Path 28:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=80
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	194	194
Object	strcat	strcat

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
194.        strcat(recovery_path, "/");
```

Dangerous Functions\Path 29:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=81
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	197	197
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
197.        strcat(boot_path, "/boot");
```

Dangerous Functions\Path 30:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=82
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	201	201
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
201.         strcat(boot_path, "/");
```

Dangerous Functions\Path 31:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=83
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	205	205
Object	strcat	strcat

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
205.         strcat(recoveryimg, "recovery.img");
```

Dangerous Functions\Path 32:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=84
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	207	207
Object	strcat	strcat

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
207.        strcat(bootimg, "boot.img");
```

Dangerous Functions\Path 33:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=85
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	209	209
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
209.        strcat(tempimg, "/temp.img");
```

Dangerous Functions\Path 34:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=86
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	213	213
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
213.        strcat(exec, recoveryimg);
```

Dangerous Functions\Path 35:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=87
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	224	224
Object	strcat	strcat

Code Snippet	
File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
224.        strcat(exec, tempimg);
```

Dangerous Functions\Path 36:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=88
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	240	240
Object	strcat	strcat

Code Snippet	
File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
240.                strcat(exec, tempimg);
```

Dangerous Functions\Path 37:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=89
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	241	241
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
241.                strcat(exec, " ");
```

Dangerous Functions\Path 38:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=90
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	242	242
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
242.                strcat(exec, bootimg);
```

Dangerous Functions\Path 39:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=91
Status	New

The dangerous function, strcat, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	260	260
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
260.          strcat(exec, recoveryimg);
```

Dangerous Functions\Path 40:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=92
Status	New

The dangerous function, strcat, was found in use at line 278 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	283	283
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void restore_original_boot(int no_flash) {

```
....
283.        strcat(boot_path, device_id);
```

Dangerous Functions\Path 41:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=93
Status	New

The dangerous function, strcat, was found in use at line 278 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	284	284
Object	strcat	strcat

Code Snippet	
File Name	android_bootable_recovery/htcdumlock.c
Method	void restore_original_boot(int no_flash) {

```
....
284.        strcat(boot_path, "/boot/");
```

Dangerous Functions\Path 42:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=94
Status	New

The dangerous function, strcat, was found in use at line 278 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	286	286
Object	strcat	strcat

Code Snippet	
File Name	android_bootable_recovery/htcdumlock.c
Method	void restore_original_boot(int no_flash) {

```
....
286.        strcat(exec, boot_path);
```

Dangerous Functions\Path 43:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=95
Status	New

The dangerous function, strcat, was found in use at line 278 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	287	287
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void restore_original_boot(int no_flash) {

```
....
287.        strcat(exec, "boot.img");
```

Dangerous Functions\Path 44:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=96
Status	New

The dangerous function, strcat, was found in use at line 64 in android_bootable_recovery/main.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	92	92
Object	strcat	strcat

Code Snippet
File Name android_bootable_recovery/main.c
Method static void dirck(struct exfat* ef, const char* path)

```
....
92.    strcat(entry_path, "/");
```

Dangerous Functions\Path 45:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=97
Status	New

The dangerous function, strcpy, was found in use at line 768 in android_bootable_recovery/glob.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	775	775
Object	strcpy	strcpy

Code Snippet
File Name android_bootable_recovery/glob.c
Method g_opendir(str, pglob)

```
....
775.                strcpy(buf, ".");
```

Dangerous Functions\Path 46:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=98
Status	New

The dangerous function, strcpy, was found in use at line 68 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	92	92
Object	strcpy	strcpy

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
92.    strcpy(device_id, "serialno");
```

Dangerous Functions\Path 47:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=99
Status	New

The dangerous function, strcpy, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	183	183
Object	strcpy	strcpy

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
183.        strcpy(twrp_device_path, "/sdcard/TWRP/htcdumlock/");
```

Dangerous Functions\Path 48:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=100
Status	New

The dangerous function, strcpy, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	189	189
Object	strcpy	strcpy

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
189.        strcpy(recovery_path, twrp_device_path);
```

Dangerous Functions\Path 49:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=101
Status	New

The dangerous function, strcpy, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	196	196
Object	strcpy	strcpy

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
196.        strcpy(boot_path, twrp_device_path);
```

Dangerous Functions\Path 50:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=102
Status	New

The dangerous function, strcpy, was found in use at line 168 in android_bootable_recovery/htcdumlock.c file. Such functions may expose information and allow an attacker to get full control over the host machine.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	204	204
Object	strcpy	strcpy

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
204.        strcpy(recoveryimg, recovery_path);
```

Memory Leak

Query Path:
CPP\Cx\CPP Medium Threat\Memory Leak Version:1

Categories

NIST SP 800-53: SC-5 Denial of Service Protection (P1)

Description

Memory Leak\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=141
Status	New

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c
Line	389	389
Object	s	s

Code Snippet
File Name   android_bootable_recovery/sockets.c
Method   asocket *create_local_socket(int fd)

```
....
389.      asocket *s = calloc(1, sizeof(asocket));
```

Memory Leak\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=142
Status	New

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c
Line	484	484
Object	s	s

Code Snippet
File Name   android_bootable_recovery/sockets.c
Method   asocket *create_remote_socket(unsigned id, atransport *t)

```
....
484.        asocket *s = calloc(1, sizeof(aremotesocket));
```

Memory Leak\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=143
Status	New

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c
Line	708	708
Object	s	s

Code Snippet

File Name	android_bootable_recovery/sockets.c
Method	asocket *create_smart_socket(void (*action_cb)(asocket *s, const char *act))

```
....
708.        asocket *s = calloc(1, sizeof(asocket));
```

Memory Leak\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=144
Status	New

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	824	824
Object	mi	mi

Code Snippet

File Name	android_bootable_recovery/ask.c
Method	int fdisk_ask_menu_add_item(struct fdisk_ask *ask, int key,

```
....
824.          mi = calloc(1, sizeof(*mi));
```

Memory Leak\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-

	BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=145
Status	New

	Source	Destination
File	android_bootable_recovery/fileutils.c	android_bootable_recovery/fileutils.c
Line	107	107
Object	p	p

Code Snippet
File Name android_bootable_recovery/fileutils.c
Method int mkdir_p(const char *path, mode_t mode)

```
....
107.         dir = p = strdup(path);
```

Memory Leak\Path 6:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=146
Status	New

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	671	671
Object	pathv	pathv

Code Snippet
File Name android_bootable_recovery/glob.c
Method globextend(path, pglob)

```
....
671.         pathv = pglob->gl_pathv ?
```

Memory Leak\Path 7:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=147
Status	New

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	690	690

Object	copy	copy

Code Snippet

File Name android_bootable_recovery/glob.c

Method globextend(path, pglob)

```
....
690.         if ((copy = malloc(p - path)) != NULL) {
```

Memory Leak\Path 8:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=148
Status	New

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	206	206
Object	buf	buf

Code Snippet

File Name android_bootable_recovery/mbsalign.c

Method char *mbs_safe_encode(const char *s, size_t *width)

```
....
206.         buf = malloc(mbs_safe_encode_size(sz));
```

Memory Leak\Path 9:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=149
Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	175	175
Object	d	d

Code Snippet

File Name android_bootable_recovery/twrpTar.cpp

Method int twrpTar::createTarFork(pid_t *tar_fork_pid) {

```
....
175.                    d = opendir(tardir.c_str());
```

Memory Leak\Path 10:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=150
Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	223	223
Object	d	d

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::createTarFork(pid_t *tar_fork_pid) {

```
....
223.                    d = opendir(tardir.c_str());
```

Memory Leak\Path 11:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=151
Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	664	664
Object	d	d

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::Generate_TarList(string Path, std::vector<TarListStruct> *TarList, unsigned long long *Target_Size, unsigned *thread_id) {

```
....
664.        d = opendir(Path.c_str());
```

Wrong Size t Allocation

Query Path:
CPP\Cx\CPP Integer Overflow\Wrong Size t Allocation Version:0

Description

Wrong Size t Allocation\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=40
Status	New

The function src_size in android_bootable_recovery/mbsalign.c at line 342 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	398	398
Object	src_size	src_size

Code Snippet

File Name	android_bootable_recovery/mbsalign.c
Method	mbsalign (const char *src, char *dest, size_t dest_size,

```
....
398.            newstr = malloc (src_size);
```

Wrong Size t Allocation\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=41
Status	New

The function size in android_bootable_recovery/HashPassword.cpp at line 38 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	40	40
Object	size	size

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
40.    unsigned char* buffer = (unsigned char*)calloc(1, size);
```

Wrong Size t Allocation\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=42
Status	New

The function size in android_bootable_recovery/HashPassword.cpp at line 57 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	59	59
Object	size	size

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
59.    unsigned char* buffer = (unsigned char*)calloc(1, size);
```

Wrong Size t Allocation\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=43
Status	New

The function lSize in android_bootable_recovery/htcdumlock.c at line 104 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	123	123
Object	lSize	lSize

Code Snippet
File Name android_bootable_recovery/htcdumlock.c

Method	void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
123.        buffer = (unsigned char*)malloc(sizeof(unsigned char) *
lSize);
```

Wrong Size t Allocation\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=44
Status	New

The function src_chars in android_bootable_recovery/mbsalign.c at line 342 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	371	371
Object	src_chars	src_chars

Code Snippet	
File Name	android_bootable_recovery/mbsalign.c
Method	mbsalign (const char *src, char *dest, size_t dest_size,

```
....
371.        str_wc = malloc (src_chars * sizeof (wchar_t));
```

Wrong Size t Allocation\Path 6:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=45
Status	New

The function path_length in android_bootable_recovery/main.c at line 64 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	84	84
Object	path_length	path_length

Code Snippet	
File Name	android_bootable_recovery/main.c

Method	static void dirck(struct exfat* ef, const char* path)

```
....
84.    entry_path = malloc(path_length + 1 + UTF8_BYTES(EXFAT_NAME_MAX) +
1);
```

Wrong Size t Allocation\Path 7:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=46
Status	New

The function sz in android_bootable_recovery/mbsalign.c at line 199 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	206	206
Object	sz	sz

Code Snippet	
File Name	android_bootable_recovery/mbsalign.c
Method	char *mbs_safe_encode(const char *s, size_t *width)

```
....
206.         buf = malloc(mbs_safe_encode_size(sz));
```

Wrong Size t Allocation\Path 8:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=47
Status	New

The function mbs_safe_encode_size in android_bootable_recovery/mbsalign.c at line 199 assigns an incorrectly calculated size to a buffer, resulting in a mismatch between the value being written and the size of the buffer it is being written into.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	206	206
Object	mbs_safe_encode_size	mbs_safe_encode_size

Code Snippet	
File Name	android_bootable_recovery/mbsalign.c

Method char *mbs_safe_encode(const char *s, size_t *width)

```
....
206.        buf = malloc(mbs_safe_encode_size(sz));
```

Buffer Overflow boundcpy WrongSizeParam

Query Path:
CPP\Cx\CPP Buffer Overflow\Buffer Overflow boundcpy WrongSizeParam Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.2 - Buffer overflows
OWASP Top 10 2017: A1-Injection

Description

Buffer Overflow boundcpy WrongSizeParam\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=25
Status	New

The size of the buffer used by scan_for_ramdisk_data in char, at line 104 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that scan_for_ramdisk_data passes to char, at line 104 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	147	147
Object	char	char

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
147.        memcpy(ramdisk, p, sizeof(char) * SCAN_SIZE);
```

Buffer Overflow boundcpy WrongSizeParam\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=26
Status	New

The size of the buffer used by PersonalizedHashBinary in prefix, at line 38 of android_bootable_recovery/HashPassword.cpp, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that PersonalizedHashBinary passes to prefix, at line 38 of android_bootable_recovery/HashPassword.cpp, to overwrite the target buffer.

	Source	Destination

File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	42	42
Object	prefix	prefix

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
42.    memcpy((void*)buffer, (void*)prefix, strlen(prefix));
```

Buffer Overflow boundcpy WrongSizeParam\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=27
Status	New

The size of the buffer used by PersonalizedHash in prefix, at line 57 of android_bootable_recovery/HashPassword.cpp, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that PersonalizedHash passes to prefix, at line 57 of android_bootable_recovery/HashPassword.cpp, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	61	61
Object	prefix	prefix

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
61.    memcpy((void*)buffer, (void*)prefix, strlen(prefix));
```

Buffer Overflow boundcpy WrongSizeParam\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=28
Status	New

The size of the buffer used by PersonalizedHashBinary in key_size, at line 38 of android_bootable_recovery/HashPassword.cpp, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that PersonalizedHashBinary passes to key_size, at line 38 of android_bootable_recovery/HashPassword.cpp, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	44	44
Object	key_size	key_size

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method void* PersonalizedHashBinary(const char* prefix, const char* key, const size_t key_size) {

```
....
44.    memcpy((void*)ptr, key, key_size);
```

Buffer Overflow boundcpy WrongSizeParam\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=29
Status	New

The size of the buffer used by PersonalizedHash in key_size, at line 57 of android_bootable_recovery/HashPassword.cpp, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that PersonalizedHash passes to key_size, at line 57 of android_bootable_recovery/HashPassword.cpp, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	63	63
Object	key_size	key_size

Code Snippet
File Name android_bootable_recovery/HashPassword.cpp
Method std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
63.    memcpy((void*)ptr, key, key_size);
```

Buffer Overflow boundcpy WrongSizeParam\Path 6:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=30
Status	New

The size of the buffer used by *mbs_safe_encode_to_buffer in len, at line 114 of android_bootable_recovery/mbsalign.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that *mbs_safe_encode_to_buffer passes to len, at line 114 of android_bootable_recovery/mbsalign.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	166	166
Object	len	len

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
166.                        memcpy(r, p, len);
```

Buffer Overflow boundcpy WrongSizeParam\Path 7:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=31
Status	New

The size of the buffer used by smart_socket_enqueue in p, at line 597 of android_bootable_recovery/sockets.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that smart_socket_enqueue passes to p, at line 597 of android_bootable_recovery/sockets.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c
Line	614	614
Object	p	p

Code Snippet
File Name android_bootable_recovery/sockets.c
Method static int smart_socket_enqueue(asocket *s, apacket *p)

```
....
614.                 p->data, p->len);
```

MemoryFree on StackVariable

Query Path:
CPP\Cx\CPP Medium Threat\MemoryFree on StackVariable Version:0
Description

MemoryFree on StackVariable\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-

	BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=33
Status	New

Calling free() (line 783) on a variable that was not dynamically allocated (line 783) in file android_bootable_recovery/ask.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	792	792
Object	mi	mi

Code Snippet

File Name android_bootable_recovery/ask.c

Method static void fdisk_ask_menu_reset_items(struct fdisk_ask *ask)

```
....
792.                    free(mi);
```

MemoryFree on StackVariable\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=34
Status	New

Calling free() (line 893) on a variable that was not dynamically allocated (line 893) in file android_bootable_recovery/ask.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	907	907
Object	mesg	mesg

Code Snippet

File Name android_bootable_recovery/ask.c

Method static int do_vprint(struct fdisk_context *cxt, int errnum, int type,

```
....
907.                    free(mesg);
```

MemoryFree on StackVariable\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=35
Status	New

Calling free() (line 893) on a variable that was not dynamically allocated (line 893) in file android_bootable_recovery/ask.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	918	918
Object	mesg	mesg

Code Snippet
File Name android_bootable_recovery/ask.c
Method static int do_vprint(struct fdisk_context *cxt, int errnum, int type,

```
....
918.          free(mesg);
```

MemoryFree on StackVariable\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=36
Status	New

Calling free() (line 988) on a variable that was not dynamically allocated (line 988) in file android_bootable_recovery/ask.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	1000	1000
Object	str	str

Code Snippet
File Name android_bootable_recovery/ask.c
Method int fdisk_info_new_partition(

```
....
1000.         free(str);
```

MemoryFree on StackVariable\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=37
Status	New

Calling free() (line 28) on a variable that was not dynamically allocated (line 28) in file android_bootable_recovery/fileutils.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/fileutils.c	android_bootable_recovery/fileutils.c
Line	56	56
Object	localtmp	localtmp

Code Snippet
File Name android_bootable_recovery/fileutils.c
Method int xmkstemp(char **tmpname, char *dir)

```
....
56.         free(localtmp);
```

MemoryFree on StackVariable\Path 6:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=38
Status	New

Calling free() (line 86) on a variable that was not dynamically allocated (line 86) in file android_bootable_recovery/fileutils.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/fileutils.c	android_bootable_recovery/fileutils.c
Line	92	92
Object	tmpname	tmpname

Code Snippet
File Name android_bootable_recovery/fileutils.c
Method int main(void)

```
....
92.   free(tmpname);
```

MemoryFree on StackVariable\Path 7:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=39
Status	New

Calling free() (line 468) on a variable that was not dynamically allocated (line 468) in file android_bootable_recovery/sockets.c may result with a crash.

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c

Line	479	479
Object	s	s

Code Snippet
File Name android_bootable_recovery/sockets.c
Method static void remote_socket_disconnect(void* _s, atransport* t)

```
....
479.        free(s);
```

Stored Buffer Overflow cpycat

Query Path:
CPP\Cx\CPP Stored Vulnerabilities\Stored Buffer Overflow cpycat Version:0

Categories

NIST SP 800-53: SI-10 Information Input Validation (P1)
OWASP Top 10 2017: A1-Injection

Description

Stored Buffer Overflow cpycat\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=156
Status	New

The size of the buffer used by flash_recovery_to_boot in twrp_device_path, at line 168 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	189
Object	line	twrp_device_path

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.           fgets(line, sizeof(line), fp);
```

▼

File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
189.         strcpy(recovery_path, twrp_device_path);
```

Stored Buffer Overflow cpycat\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=157
Status	New

The size of the buffer used by flash_recovery_to_boot in twrp_device_path, at line 168 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	196
Object	line	twrp_device_path

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.          fgets(line, sizeof(line), fp);
```

File Name android_bootable_recovery/htcdumlock.c

Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
196.         strcpy(boot_path, twrp_device_path);
```

Stored Buffer Overflow cpycat\Path 3:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=158
Status	New

The size of the buffer used by flash_recovery_to_boot in device_id, at line 168 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c

Line	77	184
Object	line	device_id

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void get_device_id(void)

```
....
77.         fgets(line, sizeof(line), fp);
```

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
184.        strcat(twrp_device_path, device_id);
```

Stored Buffer Overflow cpycat\Path 4:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=159
Status	New

The size of the buffer used by restore_original_boot in device_id, at line 278 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	283
Object	line	device_id

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void get_device_id(void)

```
....
77.         fgets(line, sizeof(line), fp);
```

File Name	android_bootable_recovery/htcdumlock.c
Method	void restore_original_boot(int no_flash) {

```
....
283.        strcat(boot_path, device_id);
```

Stored Buffer Overflow cpycat\Path 5:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=160
Status	New

The size of the buffer used by restore_original_boot in boot_path, at line 278 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	286
Object	line	boot_path

Code Snippet

File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.          fgets(line, sizeof(line), fp);
```

File Name android_bootable_recovery/htcdumlock.c

Method void restore_original_boot(int no_flash) {

```
....
286.         strcat(exec, boot_path);
```

Long Overflow

Query Path:
CPP\Cx\CPP Integer Overflow\Long Overflow Version:0

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.2 - Buffer overflows
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-10 Information Input Validation (P1)

Description

Long Overflow\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=50
Status	New

A variable of a larger data type, AssignExpr, is being assigned to a smaller data type, in 509 of android_bootable_recovery/snprintf.c. This will cause a loss of data, often the significant bits of a numerical value or the sign bit.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	513	513
Object	AssignExpr	AssignExpr

Code Snippet

File Name	android_bootable_recovery/snprintf.c
Method	static long round_ (long double value)

```
....
513.     intpart = value;
```

Long Overflow\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=51
Status	New

A variable of a larger data type, AssignExpr, is being assigned to a smaller data type, in 521 of android_bootable_recovery/snprintf.c. This will cause a loss of data, often the significant bits of a numerical value or the sign bit.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	558	558
Object	AssignExpr	AssignExpr

Code Snippet

File Name	android_bootable_recovery/snprintf.c
Method	static void fmtfp (char *buffer, size_t *currlen, size_t maxlen,

```
....
558.     intpart = ufvalue;
```

Use of Uninitialized Variable

Query Path:
CPP\Cx\CPP Medium Threat\Use of Uninitialized Variable Version:0

Categories

NIST SP 800-53: SC-5 Denial of Service Protection (P1)

Description

Use of Uninitialized Variable\Path 1:

Severity	Medium
Result State	To Verify

Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=152
Status	New

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	31	115
Object	files_count	files_count

Code Snippet

File Name	android_bootable_recovery/main.c
Method	uint64_t files_count, directories_count;

```
....
31.   uint64_t files_count, directories_count;
```

File Name	android_bootable_recovery/main.c
Method	static void dirck(struct exfat* ef, const char* path)

```
....
115.                        files_count++;
```

Use of Uninitialized Variable\Path 2:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=153
Status	New

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	31	110
Object	directories_count	directories_count

Code Snippet

File Name	android_bootable_recovery/main.c
Method	uint64_t files_count, directories_count;

```
....
31.   uint64_t files_count, directories_count;
```

File Name	android_bootable_recovery/main.c
Method	static void dirck(struct exfat* ef, const char* path)

```
....
110.                    directories_count++;
```

Divide By Zero

Query Path:
CPP\Cx\CPP Medium Threat\Divide By Zero Version:1

Description

Divide By Zero\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=23
Status	New

The application performs an illegal operation in twrpTar::createTarFork, in android_bootable_recovery/twrpTar.cpp. In line 114, the program attempts to divide by core_count, which might be evaluate to 0 (zero) at time of division. This value could be a hard-coded zero value, or received from external, untrusted input core_count in twrpTar::createTarFork of android_bootable_recovery/twrpTar.cpp, at line 114.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	211	211
Object	core_count	core_count

Code Snippet	
File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::createTarFork(pid_t *tar_fork_pid) {

```
....
211.                    target_size = encrypt_size / core_count;
```

Integer Overflow

Query Path:
CPP\Cx\CPP Integer Overflow\Integer Overflow Version:0

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.2 - Buffer overflows
FISMA 2014: System And Information Integrity
NIST SP 800-53: SI-10 Information Input Validation (P1)

Description

Integer Overflow\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=49
Status	New

A variable of a larger data type, AssignExpr, is being assigned to a smaller data type, in 104 of android_bootable_recovery/snprintf.c. This will cause a loss of data, often the significant bits of a numerical value or the sign bit.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	301	301
Object	AssignExpr	AssignExpr

Code Snippet
File Name android_bootable_recovery/snprintf.c
Method static void dopr (char *buffer, size_t maxlen, const char *format, va_list args)

```
....
301.            max = maxlen; /* ie, no max */
```

Heap Inspection

Query Path:
CPP\Cx\CPP Medium Threat\Heap Inspection Version:1

Categories

OWASP Top 10 2013: A6-Sensitive Data Exposure
FISMA 2014: Media Protection
NIST SP 800-53: SC-4 Information in Shared Resources (P1)
OWASP Top 10 2017: A3-Sensitive Data Exposure

Description

Heap Inspection\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=140
Status	New

Method globtilde at line 346 of android_bootable_recovery/glob.c defines pwd, which is designated to contain user passwords. However, while plaintext passwords are later assigned to pwd, this variable is never cleared from memory.

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	352	352
Object	pwd	pwd

Code Snippet
File Name android_bootable_recovery/glob.c
Method globtilde(pattern, patbuf, patbuf_len, pglob)

```
....
352.          struct passwd *pwd;
```

Use of Zero Initialized Pointer

Query Path:
CPP\Cx\CPP Medium Threat\Use of Zero Initialized Pointer Version:1

Categories

NIST SP 800-53: SC-5 Denial of Service Protection (P1)

Description

Use of Zero Initialized Pointer\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=154
Status	New

The variable declared in str_wc at android_bootable_recovery/mbsalign.c in line 342 is not initialized when it is used by newstr at android_bootable_recovery/mbsalign.c in line 342.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	348	398
Object	str_wc	newstr

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method mbsalign (const char *src, char *dest, size_t dest_size,

```
....
348.     wchar_t *str_wc = NULL;
....
398.           newstr = malloc (src_size);
```

Stored Buffer Overflow boundcpy

Query Path:
CPP\Cx\CPP Stored Vulnerabilities\Stored Buffer Overflow boundcpy Version:1

Categories

NIST SP 800-53: SI-10 Information Input Validation (P1)
OWASP Top 10 2017: A1-Injection

Description

Stored Buffer Overflow boundcpy\Path 1:

Severity	Medium
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=155
Status	New

The size of the buffer used by sanitize_device_id in device_id, at line 51 of android_bootable_recovery/htcdumlock.c, is not properly verified before writing data to the buffer. This can enable a buffer overflow attack, using the source buffer that get_device_id passes to line, at line 68 of android_bootable_recovery/htcdumlock.c, to overwrite the target buffer.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	57
Object	line	device_id

Code Snippet
File Name   android_bootable_recovery/htcdumlock.c
Method   void get_device_id(void)

```
....
77.          fgets(line, sizeof(line), fp);
```

▼

File Name   android_bootable_recovery/htcdumlock.c
Method   void sanitize_device_id(void) {

```
....
57.    memset(device_id, 0, strlen(device_id));
```

Unchecked Return Value

Query Path:
CPP\Cx\CPP Low Visibility\Unchecked Return Value Version:1

Categories

NIST SP 800-53: SI-11 Error Handling (P2)

Description

Unchecked Return Value\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=2
Status	New

The PersonalizedHash method calls the sprintf function, at line 57 of android_bootable_recovery/HashPassword.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	72	72
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	std::string PersonalizedHash(const char* prefix, const char* key, const size_t key_size) {

```
....
72.          sprintf(hex_hash + (index * 2), "%02X", hash[index]);
```

Unchecked Return Value\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=3
Status	New

The PersonalizedHashSP800 method calls the sprintf function, at line 83 of android_bootable_recovery/HashPassword.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/HashPassword.cpp	android_bootable_recovery/HashPassword.cpp
Line	108	108
Object	sprintf	sprintf

Code Snippet

File Name	android_bootable_recovery/HashPassword.cpp
Method	std::string PersonalizedHashSP800(const char* label, const char* context, const char* key, const size_t key_size) {

```
....
108.             sprintf(hex_hash + (index * 2), "%02x",
output[index]);
```

Unchecked Return Value\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=4
Status	New

The sanitize_device_id method calls the snprintf function, at line 51 of android_bootable_recovery/htcdumlock.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	56	56

Object	snprintf	snprintf

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void sanitize_device_id(void) {

```
....
56.    snprintf(str, DEVID_MAX, "%s", device_id);
```

Unchecked Return Value\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=5
Status	New

The get_device_id method calls the fgets function, at line 68 of android_bootable_recovery/htcdumlock.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	77
Object	fgets	fgets

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.          fgets(line, sizeof(line), fp);
```

Unchecked Return Value\Path 5:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=6
Status	New

The get_device_id method calls the snprintf function, at line 68 of android_bootable_recovery/htcdumlock.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	84	84

Object	snprintf	snprintf

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
84.                          snprintf(device_id, DEVID_MAX, "%s", token);
```

Unchecked Return Value\Path 6:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=7
Status	New

The *mbs_safe_encode_to_buffer method calls the sprintf function, at line 114 of android_bootable_recovery/mbsalign.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	131	131
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
131.                         sprintf(r, "\\x%02x", (unsigned char) *p);
```

Unchecked Return Value\Path 7:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=8
Status	New

The *mbs_safe_encode_to_buffer method calls the sprintf function, at line 114 of android_bootable_recovery/mbsalign.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	151	151

Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
151.                              sprintf(r, "\\x%02x", (unsigned
char) *p);
```

Unchecked Return Value\Path 8:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=9
Status	New

The *mbs_safe_encode_to_buffer method calls the sprintf function, at line 114 of android_bootable_recovery/mbsalign.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/mbsalign.c	android_bootable_recovery/mbsalign.c
Line	161	161
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/mbsalign.c
Method char *mbs_safe_encode_to_buffer(const char *s, size_t *width, char *buf)

```
....
161.                              sprintf(r, "\\x%02x", (unsigned
char) *p);
```

Unchecked Return Value\Path 9:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=10
Status	New

The main method calls the sprintf function, at line 691 of android_bootable_recovery/snprintf.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c

Line	734	734
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/snprintf.c
Method int main (void)

```
....
734.         sprintf (buf2, fp_fmt[x], fp_nums[y]);
```

Unchecked Return Value\Path 10:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=11
Status	New

The main method calls the sprintf function, at line 691 of android_bootable_recovery/snprintf.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/snprintf.c	android_bootable_recovery/snprintf.c
Line	748	748
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/snprintf.c
Method int main (void)

```
....
748.         sprintf (buf2, int_fmt[x], int_nums[y]);
```

Unchecked Return Value\Path 11:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=12
Status	New

The sendfailmsg method calls the snprintf function, at line 33 of android_bootable_recovery/sockets.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/sockets.c	android_bootable_recovery/sockets.c

Line	39	39
Object	snprintf	snprintf

Code Snippet
File Name android_bootable_recovery/sockets.c
Method int sendfailmsg(int fd, const char *reason)

```
....
39.       snprintf(buf, sizeof buf, "FAIL%04x", len);
```

Unchecked Return Value\Path 12:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=13
Status	New

The twrpTar::extractTarFork method calls the sprintf function, at line 482 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	567	567
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::extractTarFork() {

```
....
567.                                    sprintf(actual_filename, 
temp.c_str(), i, 0);
```

Unchecked Return Value\Path 13:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=14
Status	New

The twrpTar::tarList method calls the sprintf function, at line 763 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp

Line	774	774
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::tarList(std::vector<TarListStruct> *TarList, unsigned thread_id) {

```
....
774.            sprintf(actual_filename, temp.c_str(), thread_id,
archive_count);
```

Unchecked Return Value\Path 14:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=15
Status	New

The twrpTar::tarList method calls the sprintf function, at line 763 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	812	812
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::tarList(std::vector<TarListStruct> *TarList, unsigned thread_id) {

```
....
812.                            sprintf(actual_filename,
temp.c_str(), thread_id, archive_count);
```

Unchecked Return Value\Path 15:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=16
Status	New

The twrpTar::extractMulti method calls the sprintf function, at line 853 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination

File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	858	858
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method void* twrpTar::extractMulti(void *cookie) {

```
....
858.         sprintf(actual_filename, temp.c_str(), threadTar->thread_id,
archive_count);
```

Unchecked Return Value\Path 16:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=17
Status	New

The twrpTar::extractMulti method calls the sprintf function, at line 853 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	868	868
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method void* twrpTar::extractMulti(void *cookie) {

```
....
868.               sprintf(actual_filename, temp.c_str(), threadTar-
>thread_id, archive_count);
```

Unchecked Return Value\Path 17:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=18
Status	New

The twrpTar::get_size method calls the sprintf function, at line 1466 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1481	1481
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method unsigned long long twrpTar::get_size() {

```
....
1481.            sprintf(actual_filename, temp.c_str(), thread_id,
archive_count);
```

Unchecked Return Value\Path 18:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=19
Status	New

The twrpTar::get_size method calls the sprintf function, at line 1466 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1489	1489
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method unsigned long long twrpTar::get_size() {

```
....
1489.                    sprintf(actual_filename, temp.c_str(), i,
archive_count);
```

Unchecked Return Value\Path 19:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=20
Status	New

The twrpTar::get_size method calls the sprintf function, at line 1466 of android_bootable_recovery/twrpTar.cpp. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1495	1495
Object	sprintf	sprintf

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method unsigned long long twrpTar::get_size() {

```
....
1495.                            sprintf(actual_filename,
temp.c_str(), i, archive_count);
```

Unchecked Return Value\Path 20:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=21
Status	New

The *fdisk_new_ask method calls the ask function, at line 36 of android_bootable_recovery/ask.c. However, the code does not check the return value from this function, and thus would not detect runtime errors or other unexpected states.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	38	38
Object	ask	ask

Code Snippet
File Name android_bootable_recovery/ask.c
Method struct fdisk_ask *fdisk_new_ask(void)

```
....
38.   struct fdisk_ask *ask = calloc(1, sizeof(struct fdisk_ask));
```

TOCTOU

Query Path:
CPP\Cx\CPP Low Visibility\TOCTOU Version:1

Description

TOCTOU\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=176
Status	New

The get_device_id method in android_bootable_recovery/htcdumlock.c file utilizes fopen that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	75	75
Object	fopen	fopen

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
75.    fp = fopen("/proc/cmdline", "rt");
```

TOCTOU\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=177
Status	New

The scan_for_ramdisk_data method in android_bootable_recovery/htcdumlock.c file utilizes fopen that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	111	111
Object	fopen	fopen

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
111.        pFile = fopen(filename, "rb");
```

TOCTOU\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=178
Status	New

The twrpTar::createTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1001	1001
Object	open	open

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::createTar() {

```
....
1001.                   output_fd = open(TW_ADB_BACKUP, O_WRONLY);
```

TOCTOU\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=179
Status	New

The twrpTar::createTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1110	1110
Object	open	open

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::createTar() {

```
....
1110.                   output_fd = open(TW_ADB_BACKUP, O_WRONLY);
```

TOCTOU\Path 5:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=180
Status	New

The twrpTar::openTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1137	1137
Object	open	open

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::openTar() {

```
....
1137.               input_fd = open(tarfn.c_str(), O_RDONLY |
O_LARGEFILE);
```

TOCTOU\Path 6:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=181
Status	New

The twrpTar::openTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1230	1230
Object	open	open

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::openTar() {

```
....
1230.               input_fd = open(tarfn.c_str(), O_RDONLY |
O_LARGEFILE);
```

TOCTOU\Path 7:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=182
Status	New

The twrpTar::openTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1282	1282
Object	open	open

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::openTar() {

```
....
1282.                    input_fd = open(TW_ADB_RESTORE, O_RDONLY |
O_LARGEFILE);
```

TOCTOU\Path 8:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=183
Status	New

The twrpTar::openTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1285	1285
Object	open	open

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::openTar() {

```
....
1285.                    input_fd = open(tarfn.c_str(), O_RDONLY |
O_LARGEFILE);
```

TOCTOU\Path 9:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=184

Status	New

The twrpTar::openTar method in android_bootable_recovery/twrpTar.cpp file utilizes open that is accessed by other concurrent functionality in a way that is not thread-safe, which may result in a Race Condition over this resource.

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	1333	1333
Object	open	open

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::openTar() {

```
....
1333.                   input_fd = open(TW_ADB_RESTORE, O_RDONLY);
```

Incorrect Permission Assignment For Critical Resources

Query Path:
CPP\Cx\CPP Low Visibility\Incorrect Permission Assignment For Critical Resources Version:1

Categories

FISMA 2014: Access Control
NIST SP 800-53: AC-3 Access Enforcement (P1)
OWASP Top 10 2017: A2-Broken Authentication

Description

Incorrect Permission Assignment For Critical Resources\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=168
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	75	75
Object	fp	fp

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
75.    fp = fopen("/proc/cmdline", "rt");
```

Incorrect Permission Assignment For Critical Resources\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=169
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	111	111
Object	pFile	pFile

Code Snippet

File Name android_bootable_recovery/htcdumlock.c

Method void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
111.        pFile = fopen(filename, "rb");
```

Incorrect Permission Assignment For Critical Resources\Path 3:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=170
Status	New

	Source	Destination
File	android_bootable_recovery/fileutils.c	android_bootable_recovery/fileutils.c
Line	119	119
Object	mkdir	mkdir

Code Snippet

File Name android_bootable_recovery/fileutils.c

Method int mkdir_p(const char *path, mode_t mode)

```
....
119.                    rc = mkdir(dir, mode);
```

Incorrect Permission Assignment For Critical Resources\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=171
Status	New

	Source	Destination

File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	179	179
Object	mkdir	mkdir

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
179.        mkdir("/sdcard/TWRP", 0777);
```

Incorrect Permission Assignment For Critical Resources\Path 5:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=172
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	182	182
Object	mkdir	mkdir

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
182.        mkdir("/sdcard/TWRP/htcdumlock", 0777);
```

Incorrect Permission Assignment For Critical Resources\Path 6:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=173
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	187	187
Object	mkdir	mkdir

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
187.        mkdir(twrp_device_path, 0777);
```

Incorrect Permission Assignment For Critical Resources\Path 7:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=174
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	193	193
Object	mkdir	mkdir

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
193.        mkdir(recovery_path, 0777);
```

Incorrect Permission Assignment For Critical Resources\Path 8:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=175
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	200	200
Object	mkdir	mkdir

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void flash_recovery_to_boot(int no_flash, int no_reboot) {

```
....
200.        mkdir(boot_path, 0777);
```

Improper Resource Access Authorization

Query Path:
CPP\Cx\CPP Low Visibility\Improper Resource Access Authorization Version:1

Categories

FISMA 2014: Identification And Authentication
NIST SP 800-53: AC-3 Access Enforcement (P1)
OWASP Top 10 2017: A2-Broken Authentication

Description

Improper Resource Access Authorization\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=161
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	77
Object	fgets	fgets

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.         fgets(line, sizeof(line), fp);
```

Improper Resource Access Authorization\Path 2:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=162
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	77	77
Object	line	line

Code Snippet
File Name android_bootable_recovery/htcdumlock.c
Method void get_device_id(void)

```
....
77.         fgets(line, sizeof(line), fp);
```

Improper Resource Access Authorization\Path 3:

Severity	Low

Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=163
Status	New

	Source	Destination
File	android_bootable_recovery/htcdumlock.c	android_bootable_recovery/htcdumlock.c
Line	129	129
Object	buffer	buffer

Code Snippet

File Name	android_bootable_recovery/htcdumlock.c
Method	void scan_for_ramdisk_data(char *filename, char *ramdisk) {

```
....
129.        result = fread (buffer, 1, lSize, pFile);
```

Improper Resource Access Authorization\Path 4:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=164
Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	434	434
Object	Address	Address

Code Snippet

File Name	android_bootable_recovery/twrpTar.cpp
Method	int twrpTar::createTarFork(pid_t *tar_fork_pid) {

```
....
434.             while (read(progress_pipe[0], &fs, sizeof(fs)) > 0) {
```

Improper Resource Access Authorization\Path 5:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=165
Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp

Line	634	634
Object	Address	Address

Code Snippet
File Name android_bootable_recovery/twrpTar.cpp
Method int twrpTar::extractTarFork() {

```
....
634.                    while (read(progress_pipe[0], &fs, sizeof(fs)) >
0) {
```

Improper Resource Access Authorization\Path 6:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=166
Status	New

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	133	133
Object	fprintf	fprintf

Code Snippet
File Name android_bootable_recovery/main.c
Method static void usage(const char* prog)

```
....
133.          fprintf(stderr, "Usage: %s [-V] <device>\n", prog);
```

Improper Resource Access Authorization\Path 7:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=167
Status	New

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	170	170
Object	fputs	fputs

Code Snippet
File Name android_bootable_recovery/main.c
Method int main(int argc, char* argv[])

```
....
170.        fputs("File system checking finished. ", stdout);
```

Inconsistent Implementations

Query Path:
CPP\Cx\CPP Low Visibility\Inconsistent Implementations Version:0

Description

Inconsistent Implementations\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=1
Status	New

	Source	Destination
File	android_bootable_recovery/main.c	android_bootable_recovery/main.c
Line	145	145
Object	getopt	getopt

Code Snippet

File Name	android_bootable_recovery/main.c
Method	int main(int argc, char* argv[])

```
....
145.        while ((opt = getopt(argc, argv, "V")) != -1)
```

Use of Sizeof On a Pointer Type

Query Path:
CPP\Cx\CPP Low Visibility\Use of Sizeof On a Pointer Type Version:1

Description

Use of Sizeof On a Pointer Type\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=22
Status	New

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	499	499
Object	sizeof	sizeof

Code Snippet

File Name	android_bootable_recovery/glob.c
Method	glob0(pattern, pglob)

```
....
499.                    pglob->gl_pathc - oldpathc, sizeof(char *),
compare);
```

Potential Off by One Error in Loops

Query Path:
CPP\Cx\CPP Heuristic\Potential Off by One Error in Loops Version:1

Categories

PCI DSS v3.2: PCI DSS (3.2) - 6.5.1 - Injection flaws - particularly SQL injection
NIST SP 800-53: SI-16 Memory Protection (P1)
OWASP Top 10 2017: A1-Injection

Description

Potential Off by One Error in Loops\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=24
Status	New

The buffer allocated by <= in android_bootable_recovery/glob.c at line 240 does not correctly account for the actual size of the value, resulting in an incorrect allocation that is off by one.

	Source	Destination
File	android_bootable_recovery/glob.c	android_bootable_recovery/glob.c
Line	283	283
Object	<=	<=

Code Snippet
File Name android_bootable_recovery/glob.c
Method static int globexp2(ptr, pattern, pglob, rv)

```
....
283.         for (i = 0, pl = pm = ptr; pm <= pe; pm++)
```

Unreleased Resource Leak

Query Path:
CPP\Cx\CPP Low Visibility\Unreleased Resource Leak Version:0

Categories

NIST SP 800-53: SC-5 Denial of Service Protection (P1)

Description

Unreleased Resource Leak\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=32

Status	New

	Source	Destination
File	android_bootable_recovery/twrpTar.cpp	android_bootable_recovery/twrpTar.cpp
Line	543	543
Object	tattr	tattr

Code Snippet

File Name android_bootable_recovery/twrpTar.cpp

Method int twrpTar::extractTarFork() {

```
....
543.                              if (pthread_attr_init(&tattr)) {
```

NULL Pointer Dereference

Query Path:

CPP\Cx\CPP Low Visibility\NULL Pointer Dereference Version:1

Categories

NIST SP 800-53: SC-5 Denial of Service Protection (P1)
OWASP Top 10 2017: A1-Injection

Description

NULL Pointer Dereference\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=48
Status	New

The variable declared in 0 at android_bootable_recovery/ask.c in line 325 is not initialized when it is used by num at android_bootable_recovery/ask.c in line 325.

	Source	Destination
File	android_bootable_recovery/ask.c	android_bootable_recovery/ask.c
Line	328	328
Object	0	num

Code Snippet

File Name android_bootable_recovery/ask.c

Method int fdisk_ask_number_set_relative(struct fdisk_ask *ask, int relative)

```
....
328.         ask->data.num.relative = relative ? 1 : 0;
```

Insecure Temporary File

Query Path:

CPP\Cx\CPP Low Visibility\Insecure Temporary File Version:0

Categories

NIST SP 800-53: SC-4 Information in Shared Resources (P1)
OWASP Top 10 2017: A3-Sensitive Data Exposure

Description

Insecure Temporary File\Path 1:

Severity	Low
Result State	To Verify
Online Results	http://WIN-BA8RD5TJ8IG/CxWebClient/ViewerMain.aspx?scanid=1050083&projectid=50073&pathid=52
Status	New

	Source	Destination
File	android_bootable_recovery/fileutils.c	android_bootable_recovery/fileutils.c
Line	53	53
Object	mkstemp	mkstemp

Code Snippet
File Name android_bootable_recovery/fileutils.c
Method int xmkstemp(char **tmpname, char *dir)

```
....
53.    fd = mkstemp(localtmp);
```

Divide By Zero

Risk

What might happen

When a program divides a number by zero, an exception will be raised. If this exception is not handled by the application, unexpected results may occur, including crashing the application. This can be considered a DoS (Denial of Service) attack, if an external user has control of the value of the denominator or can cause this error to occur.

Cause

How does it happen

The program receives an unexpected value, and uses it for division without filtering, validation, or verifying that the value is not zero. The application does not explicitly handle this error or prevent division by zero from occuring.

General Recommendations

How to avoid it

- Before dividing by an unknown value, validate the number and explicitly ensure it does not evaluate to zero.
- Validate all untrusted input from all sources, in particular verifying that it is not zero before dividing with it.

- Verify output of methods, calculations, dictionary lookups, and so on, and ensure it is not zero before dividing with the result.
- Ensure divide-by-zero errors are caught and handled appropriately.

Source Code Examples

Java

Divide by Zero

```
public float getAverage(HttpServletRequest req) {
      int total = Integer.parseInt(req.getParameter("total"));
      int count = Integer.parseInt(req.getParameter("count"));

      return total / count;
}
```

Checked Division

```
public float getAverage(HttpServletRequest req) {
      int total = Integer.parseInt(req.getParameter("total"));
      int count = Integer.parseInt(req.getParameter("count"));

      if (count > 0)
             return total / count;
      else
             return 0;
}
```

Buffer Overflow boundcpy WrongSizeParam

Risk

What might happen

Buffer overflow attacks, in their various forms, could allow an attacker to control certain areas of memory. Typically, this is used to overwrite data on the stack necessary for the program to function properly, such as code and memory addresses, though other forms of this attack exist. Exploiting this vulnerability can generally lead to system crashes, infinite loops, or even execution of arbitrary code.

Cause

How does it happen

Buffer Overflows can manifest in numerous different variations. In it's most basic form, the attack controls a buffer, which is then copied to a smaller buffer without size verification. Because the attacker's source buffer is larger than the program's target buffer, the attacker's data overwrites whatever is next on the stack, allowing the attacker to control program structures.

Alternatively, the vulnerability could be the result of improper bounds checking; exposing internal memory addresses outside of their valid scope; allowing the attacker to control the size of the target buffer; or various other forms.

General Recommendations

How to avoid it

- Always perform proper bounds checking before copying buffers or strings.
- Prefer to use safer functions and structures, e.g. safe string classes over `char*`, `strncpy` over `strcpy`, and so on.
- Consistently apply tests for the size of buffers.
- Do not return variable addresses outside the scope of their variables.

Source Code Examples

MemoryFree on StackVariable

Risk

What might happen

Undefined Behavior may result with a crash. Crashes may give an attacker valuable information about the system and the program internals. Furthermore, it may leave unprotected files (e.g memory) that may be exploited.

Cause

How does it happen

Calling free() on a variable that was not dynamically allocated (e.g. malloc) will result with an Undefined Behavior.

General Recommendations

How to avoid it

Use free() only on dynamically allocated variables in order to prevent unexpected behavior from the compiler.

Source Code Examples

CPP

Bad - Calling free() on a static variable

```
void clean_up(){
  char temp[256];
  do_something();
  free(tmp);
  return;
}
```

Good - Calling free() only on variables that were dynamically allocated

```
void clean_up(){
  char *buff;
  buff = (char*) malloc(1024);
  free(buff);
  return;
}
```

Wrong Size t Allocation

Risk

What might happen

Incorrect allocation of memory may result in unexpected behavior by either overwriting sections of memory with unexpected values. Under certain conditions where both an incorrect allocation of memory and the values being written can be controlled by an attacker, such an issue may result in execution of malicious code.

Cause

How does it happen

Some memory allocation functions require a size value to be provided as a parameter. The allocated size should be derived from the provided value, by providing the length value of the intended source, multiplied by the size of that length. Failure to perform the correct arithmetic to obtain the exact size of the value will likely result in the source overflowing its destination.

General Recommendations

How to avoid it

- Always perform the correct arithmetic to determine size.
- Specifically for memory allocation, calculate the allocation size from the allocation source:
    - Derive the size value from the length of intended source to determine the amount of units to be processed.
    - Always programmatically consider the size of the each unit and their conversion to memory units - for example, by using sizeof() on the unit's type.
    - Memory allocation should be a multiplication of the amount of units being written, times the size of each unit.

Source Code Examples

CPP

Allocating and Assigning Memory without Sizeof Arithmetic

```
int *ptr;
ptr = (int*)malloc(5);
for (int i = 0; i < 5; i++)
{
      ptr[i] = i * 2 + 1;
}
```

Allocating and Assigning Memory with Sizeof Arithmetic

```
int *ptr;
ptr = (int*)malloc(5 * sizeof(int));
```

```
for (int i = 0; i < 5; i++)
{
      ptr[i] = i * 2 + 1;
}
```

Incorrect Arithmetic of Multi-Byte String Allocation

```
wchar_t * dest;
dest = (wchar_t *)malloc(wcslen(source) + 1); // Would not crash for a short "source"
wcscpy((wchar_t *)dest, source);
wprintf(L"Dest: %s\r\n", dest);
```

Correct Arithmetic of Multi-Byte String Allocation

```
wchar_t * dest;
dest = (wchar_t *)malloc((wcslen(source) + 1) * sizeof(wchar_t));
wcscpy((wchar_t *)dest, source);
wprintf(L"Dest: %s\r\n", dest);
```

Integer Overflow

Risk

What might happen

Assigning large data types into smaller data types, without proper checks and explicit casting, will lead to undefined behavior and unintentional effects, such as data corruption (e.g. value wraparound, wherein maximum values become minimum values); system crashes; infinite loops; logic errors, such as bypassing of security mechanisms; or even buffer overflows leading to arbitrary code execution.

Cause

How does it happen

This flaw can occur when implicitly casting numerical data types of a larger size, into a variable with a data type of a smaller size. This forces the program to discard some bits of information from the number. Depending on how the numerical data types are stored in memory, this is often the bits with the highest value, causing substantial corruption of the stored number. Alternatively, the sign bit of a signed integer could be lost, completely reversing the intention of the number.

General Recommendations

How to avoid it

- Avoid casting larger data types to smaller types.
- Prefer promoting the target variable to a large enough data type.
- If downcasting is necessary, always check that values are valid and in range of the target type, before casting

Source Code Examples

CPP

Unsafe Downsize Casting

```
int unsafe_addition(short op1, int op2) {

    // op2 gets forced from int into a short
    short total = op1 + op2;

    return total;
}
```

Safer Use of Proper Data Types

```
int safe_addition(short op1, int op2) {

    // total variable is of type int, the largest type that is needed
    int total = 0;

    // check if total will overflow available integer size
    if (INT_MAX - abs(op2) > op1)
    {
        total = op1 + op2;
    }
    else
    {
        // instead of overflow, saturate (but this is not always a good thing)
        total = INT_MAX
    }

    return total;
}
```

Long Overflow

Risk

What might happen

Assigning large data types into smaller data types, without proper checks and explicit casting, will lead to undefined behavior and unintentional effects, such as data corruption (e.g. value wraparound, wherein maximum values become minimum values); system crashes; infinite loops; logic errors, such as bypassing of security mechanisms; or even buffer overflows leading to arbitrary code execution.

Cause

How does it happen

This flaw can occur when implicitly casting numerical data types of a larger size, into a variable with a data type of a smaller size. This forces the program to discard some bits of information from the number. Depending on how the numerical data types are stored in memory, this is often the bits with the highest value, causing substantial corruption of the stored number. Alternatively, the sign bit of a signed integer could be lost, completely reversing the intention of the number.

General Recommendations

How to avoid it

- Avoid casting larger data types to smaller types.
- Prefer promoting the target variable to a large enough data type.
- If downcasting is necessary, always check that values are valid and in range of the target type, before casting

Source Code Examples

Dangerous Functions

Risk

What might happen

Use of dangerous functions may expose varying risks associated with each particular function, with potential impact of improper usage of these functions varying significantly. The presence of such functions indicates a flaw in code maintenance policies and adherence to secure coding practices, in a way that has allowed introducing known dangerous code into the application.

Cause

How does it happen

A dangerous function has been identified within the code. Functions are often deemed dangerous to use for numerous reasons, as there are different sets of vulnerabilities associated with usage of such functions. For example, some string copy and concatenation functions are vulnerable to Buffer Overflow, Memory Disclosure, Denial of Service and more. Use of these functions is not recommended.

General Recommendations

How to avoid it

- Deploy a secure and recommended alternative to any functions that were identified as dangerous.
    - If no secure alternative is found, conduct further researching and testing to identify whether current usage successfully sanitizes and verifies values, and thus successfully avoids the use-cases for whom the function is indeed dangerous
- Conduct a periodical review of methods that are in use, to ensure that all external libraries and built-in functions are up-to-date and whose use has not been excluded from best secure coding practices.

Source Code Examples

CPP

Buffer Overflow in gets()

```
int main()
{
       char buf[10];

       printf("Please enter your name: ");
       gets(buf); // veryveryverylongname
       if (buf == ACCEPTED_NAME)
       {
               // Do something
       }
       return 0;
}
```

Safe reading from user

```
int main()
{
      char buf[10];

      printf("Please enter your name: ");
      fgets(buf, sizeof(buf), stdin); //setting the amount of bytes to read
      if (buf == ACCEPTED_NAME)
      {
             //Do something
      }
      return 0;
}
```

Unsafe function for string copy

```
int main(int argc, char* argv[])
{
      char buf[10];
      strcpy(buf, argv[1]); // overflow occurs when len(argv[1]) > 10 bytes

      return 0;
}
```

Safe string copy

```
int main(int argc, char* argv[])
{
      char buf[10];
      strncpy(buf, argv[1], sizeof(buf));
      buf[9]= '\0'; //strncpy doesn't NULL terminates

      return 0;
}
```

Unsafe format string

```
int main(int argc, char* argv[])
{
      printf(argv[1]); // If argv[1] contains a format token, such as %s,%x or %d, will cause
an access violation
      return 0;
}
```

Safe format string

```
int main(int argc, char* argv[])
{
      printf("%s", argv[1]); // Second parameter is not a formattable string

      return 0;
}
```

Heap Inspection

Risk

What might happen

All variables stored by the application in unencrypted memory can potentially be retrieved by an unauthorized user, with privlieged access to the machine. For example, a privileged attacker could attach a debugger to the running process, or retrieve the process's memory from the swapfile or crash dump file.
Once the attacker finds the user passwords in memory, these can be reused to easily impersonate the user to the system.

Cause

How does it happen

String variables are immutable - in other words, once a string variable is assigned, its value cannot be changed or removed. Thus, these strings may remain around in memory, possibly in multiple locations, for an indefinite period of time until the garbage collector happens to remove it. Sensitive data, such as passwords, will remain exposed in memory as plaintext with no control over their lifetime.

General Recommendations

How to avoid it

Generic Guidance:

- Do not store senstiive data, such as passwords or encryption keys, in memory in plaintext, even for a short period of time.
- Prefer to use specialized classes that store encrypted memory.
- Alternatively, store secrets temporarily in mutable data types, such as byte arrays, and then promptly zeroize the memory locations.

Specific Recommendations - Java:

- Instead of storing passwords in immutable strings, prefer to use an encrypted memory object, such as SealedObject.

Specific Recommendations - .NET:

- Instead of storing passwords in immutable strings, prefer to use an encrypted memory object, such as SecureString or ProtectedData.

Source Code Examples

Java

Plaintext Password in Immutable String

```
class Heap_Inspection
{
  private string password;
  void setPassword()
 {
      password = System.console().readLine("Enter your password: ");
  }
}
```

Password Protected in Memory

```
class Heap_Inspection_Fixed
{
  private SealedObject password;

  void setPassword()
 {
      byte[] sKey = getKeyFromConfig();
      Cipher c = Cipher.getInstance("AES");
      c.init(Cipher.ENCRYPT_MODE, sKey);

      char[] input = System.console().readPassword("Enter your password: ");
      password = new SealedObject(Arrays.asList(input), c);

    //Zero out the possible password, for security.
    Arrays.fill(password, '0');
  }
}
```

CPP

Vulnerable C code

```
/* Vulnerable to heap inspection */

#include <stdio.h>


void somefunc(){
      printf("Yea, I'm just being called for the heap of it..\n");
}

void authfunc(){
        char* password = (char *) malloc(256);
        char ch;
        ssize_t k;
            int i=0;
        while(k = read(0, &ch, 1) > 0)
        {
                if (ch == '\n'){
                        password[i]='\0';
                        break;
                } else{
                        password[i++]=ch;
                        fflush(0);
                }
        }
        printf("Password: %s\n",&password[0]);
}
```

```
int main()

{

    printf("Please enter a password:\n");

      authfunc();
      printf("You can now dump memory to find this password!");
      somefunc();
      gets();

}
```

Safe C code

```
/* Pesumably safe heap */

#include <stdio.h>
#include <string.h>

#define STDIN_FILENO 0

void somefunc(){
        printf("Yea, I'm just being called for the heap of it..\n");
}

void authfunc(){
      char* password = (char*) malloc(256);
      int i=0;
      char ch;
      ssize_t k;
      while(k = read(STDIN_FILENO, &ch, 1) > 0)
      {
             if (ch == '\n'){
                   password[i]='\0';
                   break;
             } else{
                   password[i++]=ch;
                   fflush(0);
             }
      }
      i=0;
      memset(password,'\0',256);
}

int main()

{

      printf("Please enter a password:\n");
      authfunc();
      somefunc();
      char ch;
      while(read(STDIN_FILENO, &ch, 1) > 0)
      {
             if (ch == '\n')
                   break;
      }
}
```

Failure to Release Memory Before Removing Last Reference ('Memory Leak')

Weakness ID: 401 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The software does not sufficiently track and release allocated memory after it has been used, which slowly consumes remaining memory.

Extended Description

This is often triggered by improper handling of malformed data or unexpectedly interrupted sessions.

Terminology Notes

"memory leak" has sometimes been used to describe other kinds of issues, e.g. for information leaks in which the contents of memory are inadvertently leaked (CVE-2003-0400 is one such example of this terminology conflict).

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

C

C++

Modes of Introduction

Memory leaks have two common and sometimes overlapping causes:

- Error conditions and other exceptional circumstances
- Confusion over which part of the program is responsible for freeing the memory

Common Consequences

Scope	Effect
Availability	Most memory leaks result in general software reliability problems, but if an attacker can intentionally trigger a memory leak, the attacker might be able to launch a denial of service attack (by crashing or hanging the program) or take advantage of other unexpected program behavior resulting from a low memory condition.

Likelihood of Exploit

Medium

Demonstrative Examples

Example 1

The following C function leaks a block of allocated memory if the call to read() fails to return the expected number of bytes:

(Bad Code)

Example Language: **C**

```
char* getBlock(int fd) {
char* buf = (char*) malloc(BLOCK_SIZE);
if (!buf) {
return NULL;
}
if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {

return NULL;
}
return buf;
}
```

Example 2

Here the problem is that every time a connection is made, more memory is allocated. So if one just opened up more and more connections, eventually the machine would run out of memory.

(Bad Code)

Example Language: **C**

```
bar connection(){
foo = malloc(1024);
return foo;
}
endConnection(bar foo) {

free(foo);
}
int main() {

while(1) //thread 1
//On a connection
foo=connection(); //thread 2
//When the connection ends
endConnection(foo)
}
```

Observed Examples

Reference	Description
CVE-2005-3119	Memory leak because function does not free() an element of a data structure.
CVE-2004-0427	Memory leak when counter variable is not decremented.
CVE-2002-0574	Memory leak when counter variable is not decremented.
CVE-2005-3181	Kernel uses wrong function to release a data structure, preventing data from being properly tracked by other code.
CVE-2004-0222	Memory leak via unknown manipulations as part of protocol test suite.
CVE-2001-0136	Memory leak via a series of the same command.

Potential Mitigations

Pre-design: Use a language or compiler that performs automatic bounds checking.

Phase: Architecture and Design

Use an abstraction library to abstract away risky APIs. Not a complete solution.

Pre-design through Build: The Boehm-Demers-Weiser Garbage Collector or valgrind can be used to detect leaks in code. This is not a complete solution as it is not 100% effective.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Category	399	Resource Management Errors	**Development Concepts (primary)699**
ChildOf	Category	633	Weaknesses that Affect Memory	**Resource-specific Weaknesses (primary)631**
ChildOf	Category	730	OWASP Top Ten 2004 Category A9 - Denial of Service	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ChildOf	Weakness Base	772	Missing Release of Resource after Effective	**Research Concepts (primary)1000**

			Lifetime	
MemberOf	View	630	Weaknesses Examined by SAMATE	**Weaknesses Examined by SAMATE (primary)630**
CanFollow	Weakness Class	390	Detection of Error Condition Without Action	Research Concepts1000

Relationship Notes

This is often a resultant weakness due to improper handling of malformed data or early termination of sessions.

Affected Resources

- Memory

Functional Areas

- Memory management

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Memory leak
7 Pernicious Kingdoms			Memory Leak
CLASP			Failure to deallocate data
OWASP Top Ten 2004	A9	CWE More Specific	Denial of Service

White Box Definitions

A weakness where the code path has:

1. start statement that allocates dynamically allocated memory resource

2. end statement that loses identity of the dynamically allocated memory resource creating situation where dynamically allocated memory resource is never relinquished

Where "loses" is defined through the following scenarios:

1. identity of the dynamic allocated memory resource never obtained

2. the statement assigns another value to the data element that stored the identity of the dynamically allocated memory resource and there are no aliases of that data element

3. identity of the dynamic allocated memory resource obtained but never passed on to function for memory resource release

4. the data element that stored the identity of the dynamically allocated resource has reached the end of its scope at the statement and there are no aliases of that data element

References

J. Whittaker and H. Thompson. "How to Break Software Security". Addison Wesley. 2003.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	PLOVER		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-01		KDM Analytics	External
	added/updated white box definitions		
2008-08-15		Veracode	External
	Suggested OWASP Top Ten 2004 mapping		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Relationships, Other Notes, References, Relationship Notes, Taxonomy Mappings, Terminology Notes		
2008-10-14	CWE Content Team	MITRE	Internal
	updated Description		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Other Notes		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Name		
2009-07-17	KDM Analytics		External
	Improved the White Box Definition		

2009-07-27	CWE Content Team	MITRE	Internal
	updated White Box Definitions		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Modes of Introduction, Other Notes		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Relationships		

Previous Entry Names

Change Date	Previous Entry Name
2008-04-11	Memory Leak
2009-05-27	Failure to Release Memory Before Removing Last Reference (aka 'Memory Leak')

BACK TO TOP

Use of Uninitialized Variable

Weakness ID: 457 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The code uses a variable that has not been initialized, leading to unpredictable or unintended results.

Extended Description

In some languages, such as C, an uninitialized variable contains contents of previously-used memory. An attacker can sometimes control or read these contents.

Time of Introduction

- Implementation

Applicable Platforms

Languages

C: *(Sometimes)*

C++: *(Sometimes)*

Perl: *(Often)*

All

Common Consequences

Scope	Effect
Availability Integrity	Initial variables usually contain junk, which can not be trusted for consistency. This can lead to denial of service conditions, or modify control flow in unexpected ways. In some cases, an attacker can "pre-initialize" the variable using previous actions, which might enable code execution. This can cause a race condition if a lock variable check passes when it should not.
Authorization	Strings that are not initialized are especially dangerous, since many functions expect a null at the end -- and only at the end -- of a string.

Likelihood of Exploit

High

Demonstrative Examples

Example 1

The following switch statement is intended to set the values of the variables aN and bN, but in the default case, the programmer has accidentally set the value of aN twice. As a result, bN will have an undefined value.

(Bad Code)

Example Language: **C**

```
switch (ctl) {
case -1:
aN = 0;
bN = 0;
break;
case 0:
aN = i;
bN = -i;
break;
case 1:
aN = i + NEXT_SZ;
bN = i - NEXT_SZ;
break;
default:
```

```
aN = -1;
aN = -1;
break;
}
repaint(aN, bN);
```

Most uninitialized variable issues result in general software reliability problems, but if attackers can intentionally trigger the use of an uninitialized variable, they might be able to launch a denial of service attack by crashing the program. Under the right circumstances, an attacker may be able to control the value of an uninitialized variable by affecting the values on the stack prior to the invocation of the function.

Example 2

Example Languages: **C++ and Java**

```
int foo;
void bar() {
if (foo==0)
/.../
/../
}
```

Observed Examples

Reference	Description
CVE-2008-0081	Uninitialized variable leads to code execution in popular desktop application.
CVE-2007-4682	Crafted input triggers dereference of an uninitialized object pointer.
CVE-2007-3468	Crafted audio file triggers crash when an uninitialized variable is used.
CVE-2007-2728	Uninitialized random seed variable used.

Potential Mitigations

Phase: Implementation

Assign all variables to an initial value.

Phase: Build and Compilation

Most compilers will complain about the use of uninitialized variables if warnings are turned on.

Phase: Requirements

The choice could be made to use a language that is not susceptible to these issues.

Phase: Architecture and Design

Mitigating technologies such as safe string libraries and container abstractions could be introduced.

Other Notes

Before variables are initialized, they generally contain junk data of what was left in the memory that the variable takes up. This data is very rarely useful, and it is generally advised to pre-initialize variables or set them to their first values early. If one forgets -- in the C language -- to initialize, for example a char *, many of the simple string libraries may often return incorrect results as they expect the null termination to be at the end of a string.

Stack variables in C and C++ are not initialized by default. Their initial values are determined by whatever happens to be in their location on the stack at the time the function is invoked. Programs should never use the value of an uninitialized variable. It is not uncommon for programmers to use an uninitialized variable in code that handles errors or other rare and exceptional circumstances. Uninitialized variable warnings can sometimes indicate the presence of a typographic error in the code.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Weakness Base	456	Missing Initialization	**Development Concepts (primary)699 Research Concepts**

MemberOf	View	630	Weaknesses Examined by SAMATE	**(primary)1000 Weaknesses Examined by SAMATE (primary)630**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Uninitialized variable
7 Pernicious Kingdoms			Uninitialized Variable

White Box Definitions

A weakness where the code path has:

1. start statement that defines variable
2. end statement that accesses the variable
3. the code path does not contain a statement that assigns value to the variable

References

mercy. "Exploiting Uninitialized Data". Jan 2006. < http://www.felinemenace.org/~mercy/papers/UBehavior/UBehavior.zip>.

Microsoft Security Vulnerability Research & Defense. "MS08-014 : The Case of the Uninitialized Stack Variable Vulnerability". 2008-03-11. <http://blogs.technet.com/swi/archive/2008/03/11/the-case-of-the-uninitialized-stack-variable-vulnerability.aspx>.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	CLASP		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-01		KDM Analytics	External
	added/updated white box definitions		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Description, Relationships, Observed Example, Other Notes, References, Taxonomy Mappings		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Common Consequences, Demonstrative Examples, Potential Mitigations		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		

Previous Entry Names

Change Date	Previous Entry Name
2008-04-11	Uninitialized Variable

BACK TO TOP

Use of Zero Initialized Pointer

Risk

What might happen

A null pointer dereference is likely to cause a run-time exception, a crash, or other unexpected behavior.

Cause

How does it happen

Variables which are declared without being assigned will implicitly retain a null value until they are assigned. The null value can also be explicitly set to a variable, to ensure clear out its contents. Since null is not really a value, it may not have object variables and methods, and any attempt to access contents of a null object, instead of verifying it is set beforehand, will result in a null pointer dereference exception.

General Recommendations

How to avoid it

- For any variable that is created, ensure all logic flows between declaration and use assign a non-null value to the variable first.
- Enforce null checks on any received variable or object before it is dereferenced, to ensure it does not contain a null assigned to it elsewhere.
- Consider the need to assign null values in order to overwrite initialized variables. Consider reassigning or releasing these variables instead.

Source Code Examples

CPP

Explicit NULL Dereference

```
char * input = NULL;
printf("%s", input);
```

Implicit NULL Dereference

```
char * input;
printf("%s", input);
```

Java

Explicit Null Dereference

```
Object o = null;
out.println(o.getClass());
```

Stored Buffer Overflow boundcpy

Risk

What might happen

Buffer overflow attacks, in their various forms, could allow an attacker to control certain areas of memory. Typically, this is used to overwrite data on the stack necessary for the program to function properly, such as code and memory addresses, though other forms of this attack exist. Exploiting this vulnerability can generally lead to system crashes, infinite loops, or even execution of arbitrary code.

Cause

How does it happen

Buffer Overflows can manifest in numerous different variations. In it's most basic form, the attack controls a buffer, which is then copied to a smaller buffer without size verification. Because the attacker's source buffer is larger than the program's target buffer, the attacker's data overwrites whatever is next on the stack, allowing the attacker to control program structures.

Alternatively, the vulnerability could be the result of improper bounds checking; exposing internal memory addresses outside of their valid scope; allowing the attacker to control the size of the target buffer; or various other forms.

General Recommendations

How to avoid it

- Always perform proper bounds checking before copying buffers or strings.
- Prefer to use safer functions and structures, e.g. safe string classes over `char*`, `strncpy` over `strcpy`, and so on.
- Consistently apply tests for the size of buffers.
- Do not return variable addresses outside the scope of their variables.

Source Code Examples

CPP

Overflowing Buffers

```
const int BUFFER_SIZE = 10;
char buffer[BUFFER_SIZE];

void copyStringToBuffer(char* inputString)
{
    strcpy(buffer, inputString);
}
```

Checked Buffers

```
const int BUFFER_SIZE = 10;
const int MAX_INPUT_SIZE = 256;
char buffer[BUFFER_SIZE];

void copyStringToBuffer(char* inputString)
```

```
{
    if (strnlen(inputString, MAX_INPUT_SIZE) < sizeof(buffer))
    {
        strncpy(buffer, inputString, sizeof(buffer));
    }
}
```

Stored Buffer Overflow cpycat

Risk

What might happen

Buffer overflow attacks, in their various forms, could allow an attacker to control certain areas of memory. Typically, this is used to overwrite data on the stack necessary for the program to function properly, such as code and memory addresses, though other forms of this attack exist. Exploiting this vulnerability can generally lead to system crashes, infinite loops, or even execution of arbitrary code.

Cause

How does it happen

Buffer Overflows can manifest in numerous different variations. In it's most basic form, the attack controls a buffer, which is then copied to a smaller buffer without size verification. Because the attacker's source buffer is larger than the program's target buffer, the attacker's data overwrites whatever is next on the stack, allowing the attacker to control program structures.

Alternatively, the vulnerability could be the result of improper bounds checking; exposing internal memory addresses outside of their valid scope; allowing the attacker to control the size of the target buffer; or various other forms.

General Recommendations

How to avoid it

- Always perform proper bounds checking before copying buffers or strings.
- Prefer to use safer functions and structures, e.g. safe string classes over `char*`, `strncpy` over `strcpy`, and so on.
- Consistently apply tests for the size of buffers.
- Do not return variable addresses outside the scope of their variables.

Source Code Examples

Use of Function with Inconsistent Implementations

Weakness ID: 474 *(Weakness Base)* **Status:** Draft

Description

Description Summary

The code uses a function that has inconsistent implementations across operating systems and versions, which might cause security-relevant portability problems.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

C: *(Often)*

PHP: *(Often)*

All

Potential Mitigations

Do not accept inconsistent behavior from the API specifications when the deviant behavior increase the risk level.

Other Notes

The behavior of functions in this category varies by operating system, and at times, even by operating system version. Implementation differences can include:

- Slight differences in the way parameters are interpreted leading to inconsistent results.

- Some implementations of the function carry significant security risks.

- The function might not be defined on all platforms.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Weakness Class	398	Indicator of Poor Code Quality	**Development Concepts (primary)699** **Seven Pernicious Kingdoms (primary)700** **Research Concepts (primary)1000**
ParentOf	Weakness Variant	589	Call to Non-ubiquitous API	**Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Inconsistent Implementations

Content History

Submissions

Submission Date	Submitter	Organization	Source
	7 Pernicious Kingdoms		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Potential Mitigations, Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Relationships, Other Notes, Taxonomy Mappings		

Previous Entry Names

Change Date	Previous Entry Name
2008-04-11	Inconsistent Implementations

BACK TO TOP

Unchecked Return Value

Risk

What might happen

A program that does not check function return values could cause the application to enter an undefined state. This could lead to unexpected behavior and unintended consequences, including inconsistent data, system crashes or other error-based exploits.

Cause

How does it happen

The application calls a system function, but does not receive or check the result of this funciton. These functions often return error codes in the result, or share other status codes with it's caller. The application simply ignores this result value, losing this vital information.

General Recommendations

How to avoid it

- Always check the result of any called function that returns a value, and verify the result is an expected value.

- Ensure the calling function responds to all possible return values.

- Expect runtime errors and handle them gracefully. Explicitly define a mechanism for handling unexpected errors.

Source Code Examples

CPP

Unchecked Memory Allocation

```
buff = (char*) malloc(size);
strncpy(buff, source, size);
```

Safer Memory Allocation

```
buff = (char*) malloc(size+1);
if (buff==NULL) exit(1);

strncpy(buff, source, size);
buff[size] = '\0';
```

Use of sizeof() on a Pointer Type

Weakness ID: 467 *(Weakness Variant)* **Status:** Draft

Description

Description Summary

The code calls sizeof() on a malloced pointer type, which always returns the wordsize/8. This can produce an unexpected result if the programmer intended to determine how much memory has been allocated.

Time of Introduction

- Implementation

Applicable Platforms

Languages

C

C++

Common Consequences

Scope	Effect
Integrity	This error can often cause one to allocate a buffer that is much smaller than what is needed, leading to resultant weaknesses such as buffer overflows.

Likelihood of Exploit

High

Demonstrative Examples

Example 1

Care should be taken to ensure sizeof returns the size of the data structure itself, and not the size of the pointer to the data structure.

In this example, sizeof(foo) returns the size of the pointer.

(Bad Code)

Example Languages: **C and C++**

```
double *foo;
...
foo = (double *)malloc(sizeof(foo));
```

In this example, sizeof(*foo) returns the size of the data structure and not the size of the pointer.

(Good Code)

Example Languages: **C and C++**

```
double *foo;
...
foo = (double *)malloc(sizeof(*foo));
```

Example 2

This example defines a fixed username and password. The AuthenticateUser() function is intended to accept a username and a password from an untrusted user, and check to ensure that it matches the username and password. If the username and password match, AuthenticateUser() is intended to indicate that authentication succeeded.

(Bad Code)

```
/* Ignore CWE-259 (hard-coded password) and CWE-309 (use of password system for authentication) for this example. */

char *username = "admin";
char *pass = "password";

int AuthenticateUser(char *inUser, char *inPass) {
```

```
printf("Sizeof username = %d\n", sizeof(username));
printf("Sizeof pass = %d\n", sizeof(pass));

if (strncmp(username, inUser, sizeof(username))) {
printf("Auth failure of username using sizeof\n");
return(AUTH_FAIL);
}
/* Because of CWE-467, the sizeof returns 4 on many platforms and architectures. */
if (! strncmp(pass, inPass, sizeof(pass))) {
printf("Auth success of password using sizeof\n");
return(AUTH_SUCCESS);
}
else {
printf("Auth fail of password using sizeof\n");
return(AUTH_FAIL);
}
}

int main (int argc, char **argv)
{
int authResult;

if (argc < 3) {
ExitError("Usage: Provide a username and password");
}
authResult = AuthenticateUser(argv[1], argv[2]);
if (authResult != AUTH_SUCCESS) {
ExitError("Authentication failed");
}
else {
DoAuthenticatedTask(argv[1]);
}
}
```

In AuthenticateUser(), because sizeof() is applied to a parameter with an array type, the sizeof() call might return 4 on many modern architectures. As a result, the strncmp() call only checks the first four characters of the input password, resulting in a partial comparison (CWE-187), leading to improper authentication (CWE-287).

Because of the partial comparison, any of these passwords would still cause authentication to succeed for the "admin" user:

(Attack)

```
pass5
passABCDEFGH
passWORD
```

Because only 4 characters are checked, this significantly reduces the search space for an attacker, making brute force attacks more feasible.

The same problem also applies to the username, so values such as "adminXYZ" and "administrator" will succeed for the username.

Potential Mitigations

Phase: Implementation

Use expressions such as "sizeof(*pointer)" instead of "sizeof(pointer)", unless you intend to run sizeof() on a pointer type to gain some platform independence or if you are allocating a variable on the stack.

Other Notes

The use of sizeof() on a pointer can sometimes generate useful information. An obvious case is to find out the wordsize on a platform. More often than not, the appearance of sizeof(pointer) indicates a bug.

Weakness Ordinalities

Ordinality	Description
Primary	*(where the weakness exists independent of other weaknesses)*

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	465	Pointer Issues	**Development Concepts (primary)699**
ChildOf	Weakness Class	682	Incorrect Calculation	**Research Concepts (primary)1000**
ChildOf	Category	737	CERT C Secure Coding Section 03 - Expressions (EXP)	**Weaknesses Addressed by the CERT C Secure Coding Standard (primary)734**
ChildOf	Category	740	CERT C Secure Coding Section 06 - Arrays (ARR)	Weaknesses Addressed by the CERT C Secure Coding Standard734
CanPrecede	Weakness Base	131	Incorrect Calculation of Buffer Size	Research Concepts1000

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
CLASP			Use of sizeof() on a pointer type
CERT C Secure Coding	ARR01-C		Do not apply the sizeof operator to a pointer when taking the size of an array
CERT C Secure Coding	EXP01-C		Do not take the size of a pointer to determine the size of the pointed-to type

White Box Definitions

A weakness where code path has:

1. end statement that passes an identity of a dynamically allocated memory resource to a sizeof operator
2. start statement that allocates the dynamically allocated memory resource

References

Robert Seacord. "EXP01-A. Do not take the sizeof a pointer to determine the size of a type". <https://www.securecoding.cert.org/confluence/display/seccode/EXP01-A.+Do+not+take+the+sizeof+a+pointer+to+determine+the+size+of+a+type>.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	CLASP		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-01		KDM Analytics	External
	added/updated white box definitions		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Relationships, Other Notes, Taxonomy Mappings, Weakness Ordinalities		
2008-11-24	CWE Content Team	MITRE	Internal
	updated Relationships, Taxonomy Mappings		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Relationships		

BACK TO TOP

Potential Off by One Error in Loops

Risk

What might happen

An off by one error may result in overwriting or over-reading of unintended memory; in most cases, this can result in unexpected behavior and even application crashes. In other cases, where allocation can be controlled by an attacker, a combination of variable assignment and an off by one error can result in execution of malicious code.

Cause

How does it happen

Often when designating variables to memory, a calculation error may occur when determining size or length that is off by one.

For example in loops, when allocating an array of size 2, its cells are counted as 0,1 - therefore, if a For loop iterator on the array is incorrectly set with the start condition i=0 and the continuation condition i<=2, three cells will be accessed instead of 2, and an attempt will be made to write or read cell [2], which was not originally allocated, resulting in potential corruption of memory outside the bounds of the originally assigned array.

Another example occurs when a null-byte terminated string, in the form of a character array, is copied without its terminating null-byte. Without the null-byte, the string representation is unterminated, resulting in certain functions to over-read memory as they expect the missing null terminator.

General Recommendations

How to avoid it

- Always ensure that a given iteration boundary is correct:
    - With array iterations, consider that arrays begin with cell 0 and end with cell n-1, for a size n array.
    - With character arrays and null-byte terminated string representations, consider that the null byte is required and should not be overwritten or ignored; ensure functions in use are not vulnerable to off-by-one, specifically for instances where null-bytes are automatically appended after the buffer, instead of in place of its last character.
- Where possible, use safe functions that manage memory and are not prone to off-by-one errors.

Source Code Examples

CPP

Off-By-One in For Loop

```
int *ptr;
ptr = (int*)malloc(5 * sizeof(int));
for (int i = 0; i <= 5; i++)
{
     ptr[i] = i * 2 + 1; // ptr[5] will be set, but is out of bounds
}
```

Proper Iteration in For Loop

```
int *ptr;
ptr = (int*)malloc(5 * sizeof(int));
for (int i = 0; i < 5; i++)
{
      ptr[i] = i * 2 + 1; // ptr[0-4] are well defined
}
```

Off-By-One in strncat

```
strncat(buf, input, sizeof(buf) - strlen(buf)); // actual value should be sizeof(buf)-
strlen(buf)-1 - this form will overwrite the terminating nullbyte
```

Resource Locking Problems

Category ID: 411 *(Category)* **Status:** Draft

Description

Description Summary

Weaknesses in this category are related to improper handling of locks that are used to control access to resources.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	399	Resource Management Errors	**Development Concepts (primary)699**
ParentOf	Weakness Base	412	Unrestricted Externally Accessible Lock	Development Concepts699
ParentOf	Weakness Base	413	Insufficient Resource Locking	**Development Concepts (primary)699**
ParentOf	Weakness Base	414	Missing Lock Check	**Development Concepts (primary)699**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
PLOVER			Resource Locking problems

Content History

Submissions

Submission Date	Submitter	Organization	Source
	PLOVER		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Taxonomy Mappings		

BACK TO TOP

NULL Pointer Dereference

Risk

What might happen

A null pointer dereference is likely to cause a run-time exception, a crash, or other unexpected behavior.

Cause

How does it happen

Variables which are declared without being assigned will implicitly retain a null value until they are assigned. The null value can also be explicitly set to a variable, to ensure clear out its contents. Since null is not really a value, it may not have object variables and methods, and any attempt to access contents of a null object, instead of verifying it is set beforehand, will result in a null pointer dereference exception.

General Recommendations

How to avoid it

- For any variable that is created, ensure all logic flows between declaration and use assign a non-null value to the variable first.
- Enforce null checks on any received variable or object before it is dereferenced, to ensure it does not contain a null assigned to it elsewhere.
- Consider the need to assign null values in order to overwrite initialized variables. Consider reassigning or releasing these variables instead.

Source Code Examples

Insecure Temporary File

Weakness ID: 377 *(Weakness Base)* **Status:** Incomplete

Description

Description Summary

Creating and using insecure temporary files can leave application and system data vulnerable to attack.

Time of Introduction

- Architecture and Design
- Implementation

Applicable Platforms

Languages

All

Demonstrative Examples

Example 1

The following code uses a temporary file for storing intermediate data gathered from the network before it is processed.

(Bad Code)

Example Language: **C**

```
if (tmpnam_r(filename)) {

FILE* tmp = fopen(filename,"wb+");
while((recv(sock,recvbuf,DATA_SIZE, 0) > 0)&(amt!=0)) amt = fwrite(recvbuf,1,DATA_SIZE,tmp);
}
...
```

This otherwise unremarkable code is vulnerable to a number of different attacks because it relies on an insecure method for creating temporary files. The vulnerabilities introduced by this function and others are described in the following sections. The most egregious security problems related to temporary file creation have occurred on Unix-based operating systems, but Windows applications have parallel risks. This section includes a discussion of temporary file creation on both Unix and Windows systems. Methods and behaviors can vary between systems, but the fundamental risks introduced by each are reasonably constant.

Other Notes

Applications require temporary files so frequently that many different mechanisms exist for creating them in the C Library and Windows(R) API. Most of these functions are vulnerable to various forms of attacks.

The functions designed to aid in the creation of temporary files can be broken into two groups based whether they simply provide a filename or actually open a new file. - Group 1: "Unique" Filenames: The first group of C Library and WinAPI functions designed to help with the process of creating temporary files do so by generating a unique file name for a new temporary file, which the program is then supposed to open. This group includes C Library functions like tmpnam(), tempnam(), mktemp() and their C++ equivalents prefaced with an _ (underscore) as well as the GetTempFileName() function from the Windows API. This group of functions suffers from an underlying race condition on the filename chosen. Although the functions guarantee that the filename is unique at the time it is selected, there is no mechanism to prevent another process or an attacker from creating a file with the same name after it is selected but before the application attempts to open the file. Beyond the risk of a legitimate collision caused by another call to the same function, there is a high probability that an attacker will be able to create a malicious collision because the filenames generated by these functions are not sufficiently randomized to make them difficult to guess. If a file with the selected name is created, then depending on how the file is opened the existing contents or access permissions of the file may remain intact. If the existing contents of the file are malicious in nature, an attacker may be able to inject dangerous data into the application when it reads data back from the temporary file. If an attacker pre-creates the file with relaxed access permissions, then data stored in the temporary file by the application may be accessed, modified or corrupted by an attacker. On Unix based systems an even more insidious attack is possible if the attacker pre-creates the file as a link to another important file. Then, if the application truncates or writes data to the file, it may unwittingly perform damaging operations for the attacker. This is an especially serious threat if the program operates with elevated permissions. Finally, in the best case the file will be opened with the a call to open() using the O_CREAT and O_EXCL flags or to CreateFile() using the CREATE_NEW attribute, which will fail if the file already exists and therefore prevent the types of attacks described above. However, if an attacker is able to accurately predict a sequence of temporary file names, then the application may be prevented from opening necessary temporary storage causing a denial of service (DoS) attack. This type of attack would not be difficult to mount given the small amount of randomness used in the selection of the filenames generated by these functions. - Group 2: "Unique" Files: The second group of C Library functions attempts to resolve some of the security problems related to temporary files by not only generating a unique file name, but also opening the file. This group includes C Library functions like tmpfile() and its C++ equivalents prefaced with an _ (underscore), as well as the slightly better-behaved C Library function mkstemp(). The tmpfile() style functions construct a unique filename and open it in the same way that fopen() would if passed the flags "wb+", that is, as a binary file in read/write mode. If the file already exists, tmpfile() will truncate it to size zero, possibly in an attempt to assuage the security concerns mentioned earlier regarding the race condition that exists between the selection of a supposedly unique filename and the subsequent opening of the selected file. However, this behavior clearly does not solve the function's security problems. First, an attacker can pre-create the file with relaxed access-permissions that will likely be retained by the file opened by tmpfile(). Furthermore, on Unix based systems if the attacker pre-creates the file as a link to another important file, the application may use its possibly elevated permissions to truncate that file, thereby doing damage on behalf of the attacker. Finally, if tmpfile() does create a new file, the access permissions applied to that file will vary from one operating system to another, which can leave application data vulnerable even if an attacker is unable to predict the filename to be used in advance. Finally, mkstemp() is a reasonably safe way create temporary files. It will attempt to create and open a unique file based on a filename template provided by the user combined with a series of randomly generated characters. If it is unable to create such a file, it will fail and return -1. On modern systems the file is opened using mode 0600, which means the file will be secure from tampering unless the user explicitly changes its access permissions. However, mkstemp() still suffers from the use of predictable file names and can leave an application vulnerable to denial of service attacks if an attacker causes mkstemp() to fail by predicting and pre-creating the filenames to be used.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	361	Time and State	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Category	376	Temporary File Issues	**Development Concepts (primary)699**
ChildOf	Weakness Class	668	Exposure of Resource to Wrong Sphere	**Research Concepts (primary)1000**
ParentOf	Weakness Base	378	Creation of Temporary File With Insecure Permissions	**Research Concepts (primary)1000**
ParentOf	Weakness Base	379	Creation of Temporary File in Directory with Incorrect Permissions	**Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Insecure Temporary File

References

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 23, "Creating Temporary Files Securely" Page 682. 2nd Edition. Microsoft. 2002.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	7 Pernicious Kingdoms		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Other Notes, Taxonomy Mappings		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Demonstrative Examples		
2010-02-16	CWE Content Team	MITRE	Internal
	updated References		

BACK TO TOP

Improper Access Control (Authorization)

Weakness ID: 285 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The software does not perform or incorrectly performs access control checks across all potential execution paths.

Extended Description

When access control checks are not applied consistently - or not at all - users are able to access data or perform actions that they should not be allowed to perform. This can lead to a wide range of problems, including information leaks, denial of service, and arbitrary code execution.

Alternate Terms

AuthZ:	"AuthZ" is typically used as an abbreviation of "authorization" within the web application security community. It is also distinct from "AuthC," which is an abbreviation of "authentication." The use of "Auth" as an abbreviation is discouraged, since it could be used for either authentication or authorization.

Time of Introduction

- Architecture and Design
- Implementation
- Operation

Applicable Platforms

Languages

Language-independent

Technology Classes

Web-Server: *(Often)*

Database-Server: *(Often)*

Modes of Introduction

A developer may introduce authorization weaknesses because of a lack of understanding about the underlying technologies. For example, a developer may assume that attackers cannot modify certain inputs such as headers or cookies.

Authorization weaknesses may arise when a single-user application is ported to a multi-user environment.

Common Consequences

Scope	Effect
Confidentiality	An attacker could read sensitive data, either by reading the data directly from a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to read the data.
Integrity	An attacker could modify sensitive data, either by writing the data directly to a data store that is not properly restricted, or by accessing insufficiently-protected, privileged functionality to write the data.
Integrity	An attacker could gain privileges by modifying or reading critical data directly, or by accessing insufficiently-protected, privileged functionality.

Likelihood of Exploit

High

Detection Methods

Automated Static Analysis

Automated static analysis is useful for detecting commonly-used idioms for authorization. A tool may be able to analyze related configuration files, such as .htaccess in Apache web servers, or detect the usage of commonly-used authorization libraries.

Generally, automated static analysis tools have difficulty detecting custom authorization schemes. In addition, the software's design may include some functionality that is accessible to any user and does not require an authorization check; an automated technique that detects the absence of authorization may report false positives.

Effectiveness: Limited

Automated Dynamic Analysis

Automated dynamic analysis may find many or all possible interfaces that do not require authorization, but manual analysis is required to determine if the lack of authorization violates business logic

Manual Analysis

This weakness can be detected using tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session.

Specifically, manual static analysis is useful for evaluating the correctness of custom authorization mechanisms.

Effectiveness: Moderate

These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules. However, manual efforts might not achieve desired code coverage within limited time constraints.

Demonstrative Examples

Example 1

The following program could be part of a bulletin board system that allows users to send private messages to each other. This program intends to authenticate the user before deciding whether a private message should be displayed. Assume that LookupMessageObject() ensures that the $id argument is numeric, constructs a filename based on that id, and reads the message details from that file. Also assume that the program stores all private messages for all users in the same directory.

(Bad Code)

Example Language: **Perl**

```
sub DisplayPrivateMessage {
my($id) = @_;
my $Message = LookupMessageObject($id);
print "From: " . encodeHTML($Message->{from}) . "<br>\n";
print "Subject: " . encodeHTML($Message->{subject}) . "\n";
print "<hr>\n";
print "Body: " . encodeHTML($Message->{body}) . "\n";
}

my $q = new CGI;
# For purposes of this example, assume that CWE-309 and
# CWE-523 do not apply.
if (! AuthenticateUser($q->param('username'), $q->param('password'))) {
ExitError("invalid username or password");
}

my $id = $q->param('id');
DisplayPrivateMessage($id);
```

While the program properly exits if authentication fails, it does not ensure that the message is addressed to the user. As a result, an authenticated attacker could provide any arbitrary identifier and read private messages that were intended for other users.

One way to avoid this problem would be to ensure that the "to" field in the message object matches the username of the authenticated user.

Observed Examples

Reference	Description
CVE-2009-3168	Web application does not restrict access to admin scripts, allowing authenticated users to reset administrative passwords.

CVE-2009-2960	Web application does not restrict access to admin scripts, allowing authenticated users to modify passwords of other users.
CVE-2009-3597	Web application stores database file under the web root with insufficient access control (CWE-219), allowing direct request.
CVE-2009-2282	Terminal server does not check authorization for guest access.
CVE-2009-3230	Database server does not use appropriate privileges for certain sensitive operations.
CVE-2009-2213	Gateway uses default "Allow" configuration for its authorization settings.
CVE-2009-0034	Chain: product does not properly interpret a configuration option for a system group, allowing users to gain privileges.
CVE-2008-6123	Chain: SNMP product does not properly parse a configuration option for which hosts are allowed to connect, allowing unauthorized IP addresses to connect.
CVE-2008-5027	System monitoring software allows users to bypass authorization by creating custom forms.
CVE-2008-7109	Chain: reliance on client-side security (CWE-602) allows attackers to bypass authorization using a custom client.
CVE-2008-3424	Chain: product does not properly handle wildcards in an authorization policy list, allowing unintended access.
CVE-2009-3781	Content management system does not check access permissions for private files, allowing others to view those files.
CVE-2008-4577	ACL-based protection mechanism treats negative access rights as if they are positive, allowing bypass of intended restrictions.
CVE-2008-6548	Product does not check the ACL of a page accessed using an "include" directive, allowing attackers to read unauthorized files.
CVE-2007-2925	Default ACL list for a DNS server does not set certain ACLs, allowing unauthorized DNS queries.
CVE-2006-6679	Product relies on the X-Forwarded-For HTTP header for authorization, allowing unintended access by spoofing the header.
CVE-2005-3623	OS kernel does not check for a certain privilege before setting ACLs for files.
CVE-2005-2801	Chain: file-system code performs an incorrect comparison (CWE-697), preventing defauls ACLs from being properly applied.
CVE-2001-1155	Chain: product does not properly check the result of a reverse DNS lookup because of operator precedence (CWE-783), allowing bypass of DNS-based access restrictions.

Potential Mitigations

Phase: Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully mapping roles with data and functionality. Use role-based access control (RBAC) to enforce the roles at the appropriate boundaries.

Note that this approach may not protect against horizontal authorization, i.e., it will not protect a user from attacking others with the same role.

Phase: Architecture and Design

Ensure that you perform access control checks related to your business logic. These checks may be different than the access control checks that you apply to more generic resources such as files, connections, processes, memory, and database records. For example, a database may restrict access for medical records to a specific database user, but each record might only be intended to be accessible to the patient and the patient's doctor.

Phase: Architecture and Design

Strategy: Libraries or Frameworks

Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.

For example, consider using authorization frameworks such as the JAAS Authorization Framework and the OWASP ESAPI Access Control feature.

Phase: Architecture and Design

For web applications, make sure that the access control mechanism is enforced correctly at the server side on every page. Users should not be able to access any unauthorized functionality or information by simply requesting direct access to that page.

One way to do this is to ensure that all pages containing sensitive information are not cached, and that all such pages restrict access to requests that are accompanied by an active and authenticated session token associated with a user who has the required permissions to access that page.

Phases: System Configuration; Installation

Use the access control capabilities of your operating system and server environment and define your access control lists accordingly. Use a "default deny" policy when defining these ACLs.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	254	Security Features	**Seven Pernicious Kingdoms (primary)700**
ChildOf	Weakness Class	284	Access Control (Authorization) Issues	**Development Concepts (primary)699** **Research Concepts (primary)1000**
ChildOf	Category	721	OWASP Top Ten 2007 Category A10 - Failure to Restrict URL Access	**Weaknesses in OWASP Top Ten (2007) (primary)629**
ChildOf	Category	723	OWASP Top Ten 2004 Category A2 - Broken Access Control	**Weaknesses in OWASP Top Ten (2004) (primary)711**
ChildOf	Category	753	2009 Top 25 - Porous Defenses	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
ChildOf	Category	803	2010 Top 25 - Porous Defenses	**Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)800**
ParentOf	Weakness Variant	219	Sensitive Data Under Web Root	**Research Concepts (primary)1000**
ParentOf	Weakness Base	551	Incorrect Behavior Order: Authorization Before Parsing and Canonicalization	**Development Concepts (primary)699** Research Concepts1000
ParentOf	Weakness Class	638	Failure to Use Complete Mediation	Research Concepts1000
ParentOf	Weakness Base	804	Guessable CAPTCHA	**Development Concepts (primary)699** **Research Concepts (primary)1000**

Taxonomy Mappings

Mapped Taxonomy Name	Node ID	Fit	Mapped Node Name
7 Pernicious Kingdoms			Missing Access Control
OWASP Top Ten 2007	A10	CWE More Specific	Failure to Restrict URL Access
OWASP Top Ten 2004	A2	CWE More Specific	Broken Access Control

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
1	Accessing Functionality Not Properly Constrained by ACLs	
13	Subverting Environment Variable Values	

17	Accessing, Modifying or Executing Executable Files
87	Forceful Browsing
39	Manipulating Opaque Client-based Data Tokens
45	Buffer Overflow via Symbolic Links
51	Poison Web Service Registry
59	Session Credential Falsification through Prediction
60	Reusing Session IDs (aka Session Replay)
77	Manipulating User-Controlled Variables
76	Manipulating Input to File System Calls
104	Cross Zone Scripting

References

NIST. "Role Based Access Control and Role Based Security". <http://csrc.nist.gov/groups/SNS/rbac/>.

[REF-11] M. Howard and D. LeBlanc. "Writing Secure Code". Chapter 4, "Authorization" Page 114; Chapter 6, "Determining Appropriate Access Control" Page 171. 2nd Edition. Microsoft. 2002.

Content History

Submissions

Submission Date	Submitter	Organization	Source
	7 Pernicious Kingdoms		Externally Mined

Modifications

Modification Date	Modifier	Organization	Source
2008-07-01	Eric Dalci	Cigital	External
	updated Time of Introduction		
2008-08-15		Veracode	External
	Suggested OWASP Top Ten 2004 mapping		
2008-09-08	CWE Content Team	MITRE	Internal
	updated Relationships, Other Notes, Taxonomy Mappings		
2009-01-12	CWE Content Team	MITRE	Internal
	updated Common Consequences, Description, Likelihood of Exploit, Name, Other Notes, Potential Mitigations, References, Relationships		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Description, Related Attack Patterns		
2009-07-27	CWE Content Team	MITRE	Internal
	updated Relationships		
2009-10-29	CWE Content Team	MITRE	Internal
	updated Type		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Demonstrative Examples, Detection Factors, Modes of Introduction, Observed Examples, Relationships		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Alternate Terms, Detection Factors, Potential Mitigations, References, Relationships		
2010-04-05	CWE Content Team	MITRE	Internal
	updated Potential Mitigations		

Previous Entry Names

Change Date	Previous Entry Name
2009-01-12	Missing or Inconsistent Access Control

BACK TO TOP

Incorrect Permission Assignment for Critical Resource

Weakness ID: 732 *(Weakness Class)* **Status:** Draft

Description

Description Summary

The software specifies permissions for a security-critical resource in a way that allows that resource to be read or modified by unintended actors.

Extended Description

When a resource is given a permissions setting that provides access to a wider range of actors than required, it could lead to the disclosure of sensitive information, or the modification of that resource by unintended parties. This is especially dangerous when the resource is related to program configuration, execution or sensitive user data.

Time of Introduction

- Architecture and Design
- Implementation
- Installation
- Operation

Applicable Platforms

Languages

Language-independent

Modes of Introduction

The developer may set loose permissions in order to minimize problems when the user first runs the program, then create documentation stating that permissions should be tightened. Since system administrators and users do not always read the documentation, this can result in insecure permissions being left unchanged.

The developer might make certain assumptions about the environment in which the software runs - e.g., that the software is running on a single-user system, or the software is only accessible to trusted administrators. When the software is running in a different environment, the permissions become a problem.

Common Consequences

Scope	Effect
Confidentiality	An attacker may be able to read sensitive information from the associated resource, such as credentials or configuration information stored in a file.
Integrity	An attacker may be able to modify critical properties of the associated resource to gain privileges, such as replacing a world-writable executable with a Trojan horse.
Availability	An attacker may be able to destroy or corrupt critical data in the associated resource, such as deletion of records from a database.

Likelihood of Exploit

Medium to High

Detection Methods

Automated Static Analysis

Automated static analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc. Automated techniques may be able to detect the use of library functions that modify permissions, then analyze function calls for arguments that contain potentially insecure values.

However, since the software's intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated static analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes.

When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated static analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Automated Dynamic Analysis

Automated dynamic analysis may be effective in detecting permission problems for system resources such as files, directories, shared memory, device interfaces, etc.

However, since the software's intended security policy might allow loose permissions for certain operations (such as publishing a file on a web server), automated dynamic analysis may produce some false positives - i.e., warnings that do not have any security consequences or require any code changes.

When custom permissions models are used - such as defining who can read messages in a particular forum in a bulletin board system - these can be difficult to detect using automated dynamic analysis. It may be possible to define custom signatures that identify any custom functions that implement the permission checks and assignments.

Manual Static Analysis

Manual static analysis may be effective in detecting the use of custom permissions models and functions. The code could then be examined to identifying usage of the related functions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Manual Dynamic Analysis

Manual dynamic analysis may be effective in detecting the use of custom permissions models and functions. The program could then be executed with a focus on exercising code paths that are related to the custom permissions. Then the human analyst could evaluate permission assignments in the context of the intended security model of the software.

Fuzzing

Fuzzing is not effective in detecting this weakness.

Demonstrative Examples

Example 1

The following code sets the umask of the process to 0 before creating a file and writing "Hello world" into the file.

(Bad Code)

Example Language: **C**

```
#define OUTFILE "hello.out"

umask(0);
FILE *out;
/* Ignore CWE-59 (link following) for brevity */
out = fopen(OUTFILE, "w");
if (out) {
fprintf(out, "hello world!\n");
fclose(out);
}
```

After running this program on a UNIX system, running the "ls -l" command might return the following output:

(Result)

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 hello.out
```

The "rw-rw-rw-" string indicates that the owner, group, and world (all users) can read the file and write to it.

Example 2

The following code snippet might be used as a monitor to periodically record whether a web site is alive. To ensure that the file can always be modified, the code uses chmod() to make the file world-writable.

(Bad Code)

Example Language: **Perl**

```
$fileName = "secretFile.out";

if (-e $fileName) {
chmod 0777, $fileName;
}
```

```
my $outFH;
if (! open($outFH, ">>$fileName")) {
ExitError("Couldn't append to $fileName: $!");
}
my $dateString = FormatCurrentTime();
my $status = IsHostAlive("cwe.mitre.org");
print $outFH "$dateString cwe status: $status!\n";
close($outFH);
```

The first time the program runs, it might create a new file that inherits the permissions from its environment. A file listing might look like:

(Result)

```
-rw-r--r-- 1 username 13 Nov 24 17:58 secretFile.out
```

This listing might occur when the user has a default umask of 022, which is a common setting. Depending on the nature of the file, the user might not have intended to make it readable by everyone on the system.

The next time the program runs, however - and all subsequent executions - the chmod will set the file's permissions so that the owner, group, and world (all users) can read the file and write to it:

(Result)

```
-rw-rw-rw- 1 username 13 Nov 24 17:58 secretFile.out
```

Perhaps the programmer tried to do this because a different process uses different permissions that might prevent the file from being updated.

Example 3

The following command recursively sets world-readable permissions for a directory and all of its children:

(Bad Code)

Example Language: **Shell**

```
chmod -R ugo+r DIRNAME
```

If this command is run from a program, the person calling the program might not expect that all the files under the directory will be world-readable. If the directory is expected to contain private data, this could become a security problem.

Observed Examples

Reference	Description
CVE-2009-3482	Anti-virus product sets insecure "Everyone: Full Control" permissions for files under the "Program Files" folder, allowing attackers to replace executables with Trojan horses.
CVE-2009-3897	Product creates directories with 0777 permissions at installation, allowing users to gain privileges and access a socket used for authentication.
CVE-2009-3489	Photo editor installs a service with an insecure security descriptor, allowing users to stop or start the service, or execute commands as SYSTEM.
CVE-2009-3289	Library function copies a file to a new target and uses the source file's permissions for the target, which is incorrect when the source file is a symbolic link, which typically has 0777 permissions.
CVE-2009-0115	Device driver uses world-writable permissions for a socket file, allowing attackers to inject arbitrary commands.
CVE-2009-1073	LDAP server stores a cleartext password in a world-readable file.
CVE-2009-0141	Terminal emulator creates TTY devices with world-writable permissions, allowing an attacker to write to the terminals of other users.

CVE-2008-0662	VPN product stores user credentials in a registry key with "Everyone: Full Control" permissions, allowing attackers to steal the credentials.
CVE-2008-0322	Driver installs its device interface with "Everyone: Write" permissions.
CVE-2009-3939	Driver installs a file with world-writable permissions.
CVE-2009-3611	Product changes permissions to 0777 before deleting a backup; the permissions stay insecure for subsequent backups.
CVE-2007-6033	Product creates a share with "Everyone: Full Control" permissions, allowing arbitrary program execution.
CVE-2007-5544	Product uses "Everyone: Full Control" permissions for memory-mapped files (shared memory) in inter-process communication, allowing attackers to tamper with a session.
CVE-2005-4868	Database product uses read/write permissions for everyone for its shared memory, allowing theft of credentials.
CVE-2004-1714	Security product uses "Everyone: Full Control" permissions for its configuration files.
CVE-2001-0006	"Everyone: Full Control" permissions assigned to a mutex allows users to disable network connectivity.
CVE-2002-0969	Chain: database product contains buffer overflow that is only reachable through a .ini configuration file - which has "Everyone: Full Control" permissions.

Potential Mitigations

Phase: Implementation

When using a critical resource such as a configuration file, check to see if the resource has insecure permissions (such as being modifiable by any regular user), and generate an error or even exit the software if there is a possibility that the resource could have been modified by an unauthorized party.

Phase: Architecture and Design

Divide your application into anonymous, normal, privileged, and administrative areas. Reduce the attack surface by carefully defining distinct user groups, privileges, and/or roles. Map these against data, functionality, and the related resources. Then set the permissions accordingly. This will allow you to maintain more fine-grained control over your resources.

Phases: Implementation; Installation

During program startup, explicitly set the default permissions or umask to the most restrictive setting possible. Also set the appropriate permissions during program installation. This will prevent you from inheriting insecure permissions from any user who installs or runs the program.

Phase: System Configuration

For all configuration files, executables, and libraries, make sure that they are only readable and writable by the software's administrator.

Phase: Documentation

Do not suggest insecure configuration changes in your documentation, especially if those configurations can extend to resources and other software that are outside the scope of your own software.

Phase: Installation

Do not assume that the system administrator will manually change the configuration to the settings that you recommend in the manual.

Phase: Testing

Use tools and techniques that require manual (human) analysis, such as penetration testing, threat modeling, and interactive tools that allow the tester to record and modify an active session. These may be more effective than strictly automated techniques. This is especially the case with weaknesses that are related to design and business rules.

Phase: Testing

Use monitoring tools that examine the software's process as it interacts with the operating system and the network. This technique is useful in cases when source code is unavailable, if the software was not developed by you, or if you want to verify that the build phase did not introduce any new weaknesses. Examples include debuggers that directly attach to the running process; system-call tracing utilities such as truss (Solaris) and strace (Linux); system activity monitors such as FileMon, RegMon, Process Monitor, and other Sysinternals utilities (Windows); and sniffers and protocol analyzers that monitor network traffic.

Attach the monitor to the process and watch for library functions or system calls on OS resources such as files, directories, and shared memory. Examine the arguments to these calls to infer which permissions are being used.

Note that this technique is only useful for permissions issues related to system resources. It is not likely to detect application-level business rules that are related to permissions, such as if a user of a blog system marks a post as "private," but the blog system inadvertently marks it as "public."

Phases: Testing; System Configuration

Ensure that your software runs properly under the Federal Desktop Core Configuration (FDCC) or an equivalent hardening configuration guide, which many organizations use to limit the attack surface and potential risk of deployed software.

Relationships

Nature	Type	ID	Name	View(s) this relationship pertains to
ChildOf	Category	275	Permission Issues	**Development Concepts (primary)699**
ChildOf	Weakness Class	668	Exposure of Resource to Wrong Sphere	**Research Concepts (primary)1000**
ChildOf	Category	753	2009 Top 25 - Porous Defenses	**Weaknesses in the 2009 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)750**
ChildOf	Category	803	2010 Top 25 - Porous Defenses	**Weaknesses in the 2010 CWE/SANS Top 25 Most Dangerous Programming Errors (primary)800**
RequiredBy	Compound Element: Composite	689	Permission Race Condition During Resource Copy	Research Concepts1000
ParentOf	Weakness Variant	276	Incorrect Default Permissions	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	277	Insecure Inherited Permissions	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	278	Insecure Preserved Inherited Permissions	**Research Concepts (primary)1000**
ParentOf	Weakness Variant	279	Incorrect Execution-Assigned Permissions	**Research Concepts (primary)1000**
ParentOf	Weakness Base	281	Improper Preservation of Permissions	**Research Concepts (primary)1000**

Related Attack Patterns

CAPEC-ID	Attack Pattern Name	*(CAPEC Version: 1.5)*
232	Exploitation of Privilege/Trust	
1	Accessing Functionality Not Properly Constrained by ACLs	
17	Accessing, Modifying or Executing Executable Files	
60	Reusing Session IDs (aka Session Replay)	
61	Session Fixation	
62	Cross Site Request Forgery (aka Session Riding)	
122	Exploitation of Authorization	
180	Exploiting Incorrectly Configured Access Control Security Levels	
234	Hijacking a privileged process	

References

Mark Dowd, John McDonald and Justin Schuh. "The Art of Software Security Assessment". Chapter 9, "File Permissions." Page 495.. 1st Edition. Addison Wesley. 2006.

John Viega and Gary McGraw. "Building Secure Software". Chapter 8, "Access Control." Page 194.. 1st Edition. Addison-Wesley. 2002.

Maintenance Notes

The relationships between privileges, permissions, and actors (e.g. users and groups) need further refinement within the Research view. One complication is that these concepts apply to two different pillars, related to control of resources (CWE-664) and protection mechanism failures (CWE-396).

Content History

Submissions

Submission Date	Submitter	Organization	Source
2008-09-08			Internal CWE Team
	new weakness-focused entry for Research view.		

Modifications

Modification Date	Modifier	Organization	Source
2009-01-12	CWE Content Team	MITRE	Internal
	updated Description, Likelihood of Exploit, Name, Potential Mitigations, Relationships		
2009-03-10	CWE Content Team	MITRE	Internal
	updated Potential Mitigations, Related Attack Patterns		
2009-05-27	CWE Content Team	MITRE	Internal
	updated Name		
2009-12-28	CWE Content Team	MITRE	Internal
	updated Applicable Platforms, Common Consequences, Demonstrative Examples, Detection Factors, Modes of Introduction, Observed Examples, Potential Mitigations, References		
2010-02-16	CWE Content Team	MITRE	Internal
	updated Relationships		
2010-04-05	CWE Content Team	MITRE	Internal
	updated Potential Mitigations, Related Attack Patterns		

Previous Entry Names

Change Date	Previous Entry Name
2009-01-12	Insecure Permission Assignment for Resource
2009-05-27	Insecure Permission Assignment for Critical Resource

BACK TO TOP

TOCTOU

Risk

What might happen

At best, a Race Condition may cause errors in accuracy, overidden values or unexpected behavior that may result in denial-of-service. At worst, it may allow attackers to retrieve data or bypass security processes by replaying a controllable Race Condition until it plays out in their favor.

Cause

How does it happen

Race Conditions occur when a public, single instance of a resource is used by multiple concurrent logical processes. If the these logical processes attempt to retrieve and update the resource without a timely management system, such as a lock, a Race Condition will occur.

An example for when a Race Condition occurs is a resource that may return a certain value to a process for further editing, and then updated by a second process, resulting in the original process' data no longer being valid. Once the original process edits and updates the incorrect value back into the resource, the second process' update has been overwritten and lost.

General Recommendations

How to avoid it

When sharing resources between concurrent processes across the application ensure that these resources are either thread-safe, or implement a locking mechanism to ensure expected concurrent activity.

Source Code Examples

Java

Different Threads Increment and Decrement The Same Counter Repeatedly, Resulting in a Race Condition

```
	public static int counter = 0;
	public static void start() throws InterruptedException {
		incrementCounter ic;
		decrementCounter dc;
		while(counter == 0) {
			counter = 0;
			ic = new incrementCounter();
			dc = new decrementCounter();
			ic.start();
			dc.start();
			ic.join();
			dc.join();
		}
		System.out.println(counter); //Will stop and return either -1 or 1 due to race
condition over counter
	}

	public static class incrementCounter extends Thread {
	    public void run() {
		counter++;
	    }
```

```
        }

        public static class decrementCounter extends Thread {
            public void run() {
                counter--;
            }
        }
```

Different Threads Increment and Decrement The Same Thread-Safe Counter Repeatedly, Never Resulting in a Race Condition

```
        public static int counter = 0;
        public static Object lock = new Object();

        public static void start() throws InterruptedException {
                incrementCounter ic;
                decrementCounter dc;
                while(counter == 0) { // because of proper locking, this condition is never false
                        counter = 0;
                        ic = new incrementCounter();
                        dc = new decrementCounter();
                        ic.start();
                        dc.start();
                        ic.join();
                        dc.join();
                }
                System.out.println(counter); // Never reached
        }

        public static class incrementCounter extends Thread {
            public void run() {
                synchronized (lock) {
                        counter++;
                }
            }
        }

        public static class decrementCounter extends Thread {
            public void run() {
                synchronized (lock) {
                        counter--;
                }
            }
        }
```

Scanned Languages

Language	Hash Number	Change Date
CPP	4541647240435660	6/19/2024
Common	0105849645654507	6/19/2024