What application security tools vendors don’t want you to know and holes they will never find!

Mark Curphey
John Viega
The Open Web Application Security Project (OWASP) is an open community dedicated to enabling organizations to develop, purchase, and maintain applications that can be trusted. All of the OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We advocate approaching application security as a people, process, and technology problem because the most effective approaches to application security include
McAfee to buy Foundstone for US$86 million

PAUL ROBERTS, IDG NEWS SERVICE
17/08/2004 08:20:52

Antivirus software company, McAfee, is buying Foundstone, which makes software for detecting and managing software vulnerabilities, for $US86 million in cash.

The acquisition will add Foundstone's line of vulnerability management software to McAfee's growing list of security products. McAfee plans to
How Important is Context?
#define MAXSTRLLEN(s) (sizeof(s)/sizeof(s[0]))

if (bstrURL != NULL) {
    WCHAR szTmp[MAX_PATH];
    LPCWSTR szExtSrc;
    LPWSTR szExtDst;

    wcsncpy( szTmp, bstrURL, MAXSTRLLEN(szTmp) );
    szTmp[MAXSTRLLEN(szTmp)-1] = 0;

    szExtSrc = wcsrchr( bstrURL, '.' );
    szExtDst = wcsrchr( szTmp, '.' );

    if(szExtDst) {
        szExtDst[0] = 0;

        if(IsDesktop()) {
            wcsncat( szTmp, L"__DESKTOP", MAXSTRLLEN(szTmp) );
            wcsncat( szTmp, szExtSrc , MAXSTRLLEN(szTmp) );
        }
    }
}

// rest of code snipped
#if 0
#define MAXSTRLEN(s) (sizeof(s)/sizeof(s[0]))

if (bstrURL != NULL) {
    WCHAR szTmp[MAX_PATH];
    LPCWSTR szExtSrc;
    LPWSTR szExtDst;

    wcsncpy(szTmp, bstrURL, MAXSTRLEN(szTmp));
    szTmp[MAXSTRLEN(szTmp)-1] = 0;

    szExtSrc =wcsrchr(bstrURL, '.');
    szExtDst =wcsrchr(szTmp, '.');

    if(szExtDst) {
        szExtDst[0] = 0;

        if(IsDesktop()) {
            wcsncat(szTmp, L"__DESKTOP", MAXSTRLEN(szTmp));
            wcsncat(szTmp, szExtSrc, MAXSTRLEN(szTmp));
        }
    }
}

// rest of code snipped
#endif
September 01, 2006

Examining defects in the Firefox code base

Submitted by Adam Harrsion, Klocwork

Using Klocwork’s K7 static analysis tool, I examined the large and complicated code base of the popular open source browser, Firefox. Overall it is clear that Firefox is a very well written and high quality piece of software. Several builds were performed on the code, culminating in the final analysis of version 1.5.0.6. The analysis resulted in 655 defects and 71 potential security vulnerabilities. The Firefox team has been given the analysis results, and they will determine if or how they will deal with the issues.

Only someone with in-depth knowledge and background of the Firefox code could judge the danger of a particular security vulnerability; therefore, I have not included more detailed information of these security vulnerabilities that could lead to the spreading of unfounded rumours of potential exploits. However, for those interested, I’ve provided more details of the defects below.
Well, I'm Back

September 14, 2006

Static Analysis And Scary Headlines

A few days ago Slashdot trumpeted the headline "611 Defects, 71 Vulnerabilities Found In Firefox", based on a post by Adam Harrison who had applied his company's static code analysis tool to the Firefox code. That's not an unfair summary since Harrison's post says "The analysis resulted in 655 defects and 71 potential security vulnerabilities."

The problem is Klocwork, like most other static analysis tools, reports false positives; i.e., it reports problems that are not actually bugs in the code. (More precisely, it may identify error behaviours that actually cannot occur in any run of the program.) That itself is not a problem, but when reporting the results of these tools you must make clear how many error reports the tool produced and how many of those have been verified by humans as corresponding to actual bugs which would affect some run of the program. In this case, it was not clear at all. We're
Mozilla vs. Klocwork
611 “defects”
72 “vulnerabilities”

3 verified bugs

99.5% useless?
Review: 'Hacker-In-A-Box' Tool Tests Attack Scenarios

By Mario Morejon, CRN
Wed. Aug. 23, 2006
Page 1 of 2

Few "ethical" hackers can provide simulated attacks with the level of sophistication that Cenzic offers in its Hailstorm "hacker-in-a-box" penetration tester.

Hailstorm's unique non-signature based technology interprets results during realtime attacks without comparing results with signature-based databases. The tool's interpreting engine eliminates false positives by providing generic solutions to attacks.
January 2003

WIDE OPEN ON PORT 80

How good are Web app scanners at rooting out vulnerabilities? We test two of the leading tools head-to-head to find out.

BY KELLY WHITE AND YONG-GON CHON

You're feeling pretty good about the security of your Internet-facing infrastructure. You've been diligent about vulnerability assessments and follow-up remediation to close the holes. Your last scan, using a commercial VA scanner or freeware, such as Nessus, revealed no known vulnerabilities. The only two IP addresses visible externally are your mail gateway and the load balancer for your Web servers.

Then you start thinking about the corporate sales and procurement applications that reside behind ports 80 (HTTP) and 443 (SSL). VA scanners won't touch the possible
What kind of talk is this?

Tools that try to find security holes in software

Way for us to understand and rationalize why they are so bad and unlikely to get much better soon

It is a realistic state of the union about the current state of application security technology and how it is being marketed and applied

Dr. Holger Peine
http://fhgonline.fraunhofer.de/server?suche-publica&num=048.06/D&iese
(N.B. about overly generous quote to Cenzic)

Arian Evans
1. What the customer described
2. How the project manager interpreted it
3. How the business analyst interpreted it
4. How the (expensive) business consultant saw it
5. How the developer wrote it
6. How the project was documented
7. What operations installed
8. How the consultants billed the project
9. How it was supported
10. What the customer really wanted
Implementation Bugs vs. Design Flaws
# Security Frame of Reference

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration Management</td>
<td>Configs, security managers, web server settings etc.</td>
</tr>
<tr>
<td>Authentication</td>
<td>Knowing users and entities are who they claim to be</td>
</tr>
<tr>
<td>Authorization</td>
<td>Who can do what to whom, TOCTOU etc.</td>
</tr>
<tr>
<td>Data Protection (Transit &amp; Storage)</td>
<td>Encrypted passwords, on-wire protection, channel sinks, encrypted configuration files etc.</td>
</tr>
<tr>
<td>Data Validation</td>
<td>Valid, well formed, free of malicious payloads etc.</td>
</tr>
<tr>
<td>Auditing and Logging</td>
<td>Knowing who does what to whom etc.</td>
</tr>
<tr>
<td>Error &amp; Exception Handling</td>
<td>What happens when the pooh hits the fan etc.</td>
</tr>
<tr>
<td>User Management</td>
<td>Password reset, registration, licensing etc.</td>
</tr>
</tbody>
</table>
2005-2006 Client Vulnerability Breakdown by Foundstone SecurityFrame®

- Error Handling & Exception Management: 6.05%
- Logging & Auditing: 6.94%
- Data Validation: 20.56%
- User & Session Management: 17.54%
- Configuration Management: 13.77%
- Authorization: 15.10%
- Authentication: 10.89%
- Data Protection in Storage & Transit: 9.15%
<table>
<thead>
<tr>
<th>Security Reference Frame</th>
<th>Effectiveness of Assessment Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Web App Scanners</td>
</tr>
<tr>
<td></td>
<td>Bug</td>
</tr>
<tr>
<td>Data Validation</td>
<td>![ ]</td>
</tr>
</tbody>
</table>

* unscientific, based on experience
Configuration Management

Implementation Bug

Hard coded connection string in configuration files

Use of common crypto keys across implementations

Design Flaw

ASP.NET application running in partial trust

Revert.ToSelf();

* good at many web server config issues
Web App Scanners

Data Validation

Implementation Bug

- Stored cross site scripting (even basic XSS in some cases)
- SQL injection (non ‘)
- Buffer overflows NOT HTTP 500’s!

Design Flaw

- Canonicalization
- Internationalization

* Their strongest category
Data protection

**Implementation Bug**

- Weak random number generators
- Secure memory management issues

**Design Flaw**

- Clear text passwords stored in database
- Weak algorithms *
- Reusing keys with stream ciphers
User Management

Implementation Bug
- Password generation on reset
- Weak session ID’s

Design Flaw
- Clear text passwords in the database
- Password expiry
- Password reset sent sent in clear
Grep: Lack of Context

...  
strcpy(dst, src); // Generally a “high severity” error  
...  
strncpy(dst, src); // Generally a filtered out “low sev”  
...
Grep: Lack of Context

...  
// Generally a “high severity” error
strcpy(dst, src); // Generally a “high severity” error
...

// Generally “low severity”, filtered out by default
strncpy(dst, src, n);
...

void copy_20(char *src) {
    char dst[20];
    int n;
    
    if (strlen(src) > 19) {
        return 0;
    }
    strcpy(dst, src);
    return strdup(dst);
}
void copy(char *dst, char *src) {
  int n = strlen(src);

  strncpy(dst, src, n);
  return strdup(dst);
}

char d[20];
copy(d, arbitrary_user_input);
Grep-style

• Cons:
  – 95%+ false positives for most apps
  – False negatives when rules ignore API
  – `while(i<n) buf[i++] = getc();`
  – Reports: `char crlf[]="\r\n"; strcat("foo", crlf);`

• Pros:
  – Gives manual auditor a starting point
  – Easy to support new languages
  – Immediate results on any code base
Let’s try to do better with “real” static analysis!
Sample program

```c
void main(int argc, char **argv) {
    char b1[100] = {0,}; // alloc(B1) <- 100
    char b2[100] = {0,}; // alloc(B2) <- 100
    char b3[100] = {0,}; // alloc(B3) <- 100

    strcpy(b2, b1); // len(B2)<-len(B1)<- 100; No error.
    if (argc > 1) {
        // len(B3) <- max(len(argv), 400)
        strncpy(b3, argv[1], 400); // alloc(B3) == 100.
            // len > alloc: ERROR!
    }
}
Another program

// alloc(ARGV) <- len(ARGV) <- [0,MAX]
void main(int argc, char **argv) {
  char *b1 = malloc(100);   // alloc(B1) <- [100,100]
  char *b2 = malloc(100);   // alloc(B2) <- [100,100]
  int i;

  strcpy(b2, "foo");    // len(B2) <- 4
  if (argc > 1 && strlen(argv[1]) < 100)
    strcpy(b1, argv[1]);  // len(B1) <- len(ARGV)
  for (i=0; i<3; i++)    // i <- i + 1
    strcat(b2, ".");     // len(B2) <- len(B2) + 1
}
i <- i + 1

len(ARGV) [0,max] -> len(B1) [0,0] -> len(B2) [0,0]

alloc(ARGV) [0,max] -> alloc(B1) [100,100] -> alloc(B2) [100,100]
\[
\text{alloc}(\text{ARGV}) \quad [0, \text{max}] \\
\text{alloc}(B_1) \quad [100, 100] \\
\text{alloc}(B_2) \quad [100, 100] \\
\text{len}(\text{ARGV}) \quad [0, \text{max}] \\
\text{len}(B_1) \quad [0, 0] \\
\text{len}(B_2) \quad [0, 0] \\
\]

\[
\text{i} \quad [0, 0] \\
i \leftarrow i + 1 \\
\text{len}(\text{ARGV}) \rightarrow \text{len}(B_1) \\
\text{len}(B_2) \rightarrow \text{len}(B_2) + 1 \\
\]

\[4\]
allocate(ARGV)
allocate(B1)
allocate(B2)

len(ARGV) > len(B1)
len(B2) > len(B2) + 1

i <- i + 1
alloc(ARGV) [0,max]

alloc(B1) [100,100]

len(ARGV) [0,max]

len(B1) [0,max]

len(B2) [0,1]

4

i <- i + 1

len(ARGV) [0,max] -> len(B1) [0,max] -> len(B2) [0,1] -> len(B2) + 1
alloc(ARGV) [0,max]

alloc(B1) [100,100]

alloc(B2) [100,100]

len(ARGV) [0,max]

len(B1) [0,max]

len(B2) [0,max]

len(B2) - len(ARGV) > len(B1)

len(B2) - len(ARGV) > len(B2) + 1

i <- i + 1

len(ARGV) -> len(B1)

len(B2) -> len(B2) + 1

4
\text{alloc}(\text{ARGV}) \quad \text{[0,max]}

\text{alloc}(\text{B1}) \quad \text{[100,100]}

\text{alloc}(\text{B2}) \quad \text{[100,100]}

\text{len}(\text{ARGV}) \quad \text{[0,max]}

\text{len}(\text{B1}) \quad \text{[0,max]}

\text{len}(\text{B2}) \quad \text{[0,max]}

\text{i} \quad \text{[0,0]}

\text{i} \leftarrow \text{i} + 1

\text{len}(\text{ARGV}) \quad \text{-> len}(\text{B1})

\text{len}(\text{B2}) \quad \text{-> len}(\text{B2}) + 1

4
\text{alloc}(\text{ARGV}) \quad [0, \text{max}]

\text{alloc}(\text{B1}) \quad [100, 100]

\text{alloc}(\text{B2}) \quad [100, 100]

\text{len}(\text{ARGV}) \quad [0, \text{max}]

\text{len}(\text{B1}) \quad [0, \text{max}]

\text{len}(\text{B2}) \quad [0, \text{max}]

\text{len}(\text{B2}) \rightarrow \text{len}(\text{B2}) + 1

\text{i} \quad [0, \text{max}]

\text{i} \leftarrow \text{i} + 1

\text{len}(\text{ARGV}) \rightarrow \text{len}(\text{B1})
Okay: no incoming edges to \text{len(ARGV)}

RANGE OVERLAPS: B1 MAY OVERFLOW

RANGE OVERLAPS: B2 MAY OVERFLOW
void main(int argc, char **argv) {
    char *b1 = malloc(100);
    char *b2 = malloc(100);
    int i;
    strcpy(b2, "foo");
    if (argc > 1 && strlen(argv[1]) < 100)
        strcpy(b1, argv[1]);
    for (i=0; i<3; i++)
        strcat(b2, ".");
}

Is this the b2 vuln?
void main(int argc, char **argv) {
    char *b1 = malloc(100);
    char *b2 = malloc(100);
    int i;

    strcpy(b2, "foo");
    if (argc > 1 && strlen(argv[1]) < 100)
        strcpy(b1, argv[1]);
    for (i=0; i<3; i++)
        strcat(b2, ".");
}
The Program Again

```c
void main(int argc, char **argv) {
    char *b1 = malloc(100);
    char *b2 = malloc(100);
    int i;

    strcpy(b2, "foo");
    // b1 will never be less than 100.
    if (argc > 1 && strlen(argv[1]) < 100)
        strcpy(b1, argv[1]);
    for (i=0;i<3;i++)
        strcat(b2, ".");
}
```
A good analysis requires some understanding of control flow!
Many analyses aren’t worth it!

- Over Grep:
  - No great improvement in false positives
  - Parsing code well is extremely complex
    - Perl, anyone?

- In general:
  - Capturing semantics is never-ending
  - Specify 3rd-party libraries, etc?
#define len(x) strlen(x)

void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int  i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}
```
Control Flow

```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
```

Entry
argc: [0,max]
others: nil

alloc(b): 100
len(b): 4
Control Flow

```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}
```

Entry
- argc: [0,max]
- others: nil

Write to B. Does it overflow?

alloc(b): 100
len(b): 4
Control Flow

void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}

Entry
argc: [0,max]
others: nil

alloc(b): 100
len(b): 4

No. At this node, B is alloc'd to 100, actual len of 4.
void main(int argc, char **argv) {
    char *b = malloc(100);
    int  i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}
void main(int argc, char **argv) {
    char *b = malloc(100);
    int   i;
    
    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}
void main(int argc, char **argv) {
    char *b = malloc(100);
    int  i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}
Control Flow

void main(int argc, char **argv) {
    char *b = malloc(100);
    int  i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0;i<3;i++)
        strcat(b, ".");
}

Entry
argc: [0,max]
others: nil

True
alloc(b): 100
len(b): 4

False
argc: [2, max]
len(argv): [0, 100]
Control Flow

```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
```
Control Flow

```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
```

No overflow. b is alloc'd to 100, len can be no more than 100 after null is added.
void main(int argc, char **argv) {
    char *b = malloc(100);
    int  i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
Control Flow

void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}

Use the worst case assumption for the length of b.
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
Control Flow

```c
void main(int argc, char **argv) {
    char *b = malloc(100);
    int i;

    strcpy(b, "foo");
    if(argc > 1 && len(argv[1]) < 100)
        strcpy(b, argv[1]);
    for (i=0; i<3; i++)
        strcat(b, ".");
}
```

ERROR: \texttt{len(b) > alloc(b)}!!!
Control Flow

Could show you the graph to help you debug...
Control Flow

If you're a rocket scientist 😞

(graphs get big and complex in real programs)
Control Flow

foo.c:2412: example(): Possible buffer overflow of variable dst
Stack trace:
  foo.c:1733: process_data()
  network.c:432: read_from_socket()
  main.c:94: main_loop()
  main.c:32: main()

Though, we could show you (one possible) “stack trace” instead... (far better than dynamic analysis tools!)
Control Flow

foo.c:2412: example(): Possible buffer overflow of variable dst

Data trace:
  foo.c:1733: process_data()
  network.c:432: read_from_socket()
    |    Data received from external socket

Or, we could show where the data came from
Not just memory stuff...

SQL Injection error: WebGoat/src/lessons/
    lessons.ChallengeScreen.doStage2 line 183
Source argument: query
Potential unsafe contents: *;'&\
Input source: Network Data:
    lessons.ChallengeScreen.doStage2 line 178
void main(int argc, char **argv) {
    char *b = malloc(100000);
    int n = argc;

    for (i=0;i<n;i++)
        strcat(b, ".");
}

void main(int argc, char **argv) {
    char *b = malloc(100000);
    int n = argc;

    for (i=0; i<n; i++)
        strcat(b, ".");
}

In a more complex example, would we really have to “run” the loop MAX_INT times?
void main(int argc, char **argv) {
    char *b = malloc(100000);
    int n = argc;

    for (i=0;i<n;i++)
        strcat(b, "." );
}

In this case, we could multiply the effect by the maximum value of n.
void main(int argc, char **argv) {
    char *b = malloc(100000);
    int n = argc;

    for (i=0; i<n; i++)
        strcat(b, "." );
}

More complex cases aren't that easy, and require approximations!
We lost accuracy when we merged.
Issue 2

Entry
- argc: [0, max]
- others: nil

True
- alloc(b): 100
- len(b): 4

False
- argc: [2, max]
- len(argv): [0, 100]
- len(b): [0, 100]

i = 0
- len(b): [0, 101]

i = 0
- len(b): 4
Issue 2

We can show which path is bad!
And, future calculations become much more accurate.
Issue 2

An exponential explosion of nodes
Only feasible for single functions (intraprocedural analysis)
Issue 2

Full path analysis is even less feasible when we consider exits from complex loops.
The problem with intraprocedural

```c
char *magic_function(char *a, char *b) {
    char *p1 = a;
    char *p2 = b;

    while (*p2)
        *p1++ = *p2++;
    return a;
}
```
The problem with intraprocedural

```c
char *magic_function(char *a, char *b) {
    char *p1 = a;
    char *p2 = b;

    while (*p2)
        *p1++ = *p2++;
    return a;
}
```

No matter how accurate we get inside the procedure, we are in a catch-22 (spam vs. ignore)
The problem with intraprocedural

```c
char *magic_function(char *a, char *b) {
    char *p1 = a;
    char *p2 = b;

    while (*p2)
        *p1++ = *p2++;

    return a;
}
```

Instead of erroring, we can “summarize” the generic properties.
The problem with intraprocedural

def magic_function(a, b):
    p1 = a
    p2 = b

    while p2:
        p1 = p1 + 1
        p2 = p2 + 1

    return a

e.g., len(a) <- len(b)
The problem with intraprocedural

```c
char *magic_function(char *a, char *b) {
    char *p1 = a;
    char *p2 = b;

    while (*p2)
        *p1++ = *p2++;
    return a;
}
```

Scaling algorithms to an entire program can greatly improve accuracy... and decrease efficiency!
Using environmental knowledge

- Socket vs. file
- Consider data from config files / registry
- Analyze two communicating programs together
There will always be falses

- For some things, even false negatives
  - e.g., anything in C
- Lots of things need to be approximated and are tough to approximate well
  - Arrays and pointers
  - Dynamic dispatch
  - Built in containers
- Okay, it’s an overflow, but is it exploitable?
  - Do you care?
Building good tools is hard!

• Good analysis takes years
  – Most companies haven’t bothered to try!
• Tool should handle all dev environments
  – efficiency + checkins?
• Tools should be easy enough for my mom
• Binary analysis is far, far harder!
• Few people do even a reasonable job.
secureConnect (host, port):
    s = sslConnect(gethostbyaddr(host), port)
    cert = get_cert(s)
    if ! certSignedByTrustedRoot(cert):
        raise "SSLError"
    if cert.DN <> host:
        raise "SSLError"
    if ! subjAltNameMatches(cert, host):
        raise "SSLError"
    if certRevoked(cert):
        raise "SSLError"
    return s
If you’re not an auditor, it probably isn’t cost effective!
Notes on Buying Automated Tools

Trials are limited for a reason (as are the EULA’s)

Make sure you test them on your own site / code
Basic Conclusion

“The height of mediocrity is still low”
Basic Conclusion

Accuracy on basic software today is mediocre at best

It is really easy to write an application that can’t be automatically scanned

It is really hard to write an automated scanner than can effectively analyze software
PCI Data Security Standards
6.6 Ensure that all web-facing applications are protected against known attacks by applying either of the following methods:

• Having all custom application code reviewed for common vulnerabilities by an organization that specializes in application security

• Installing an application layer firewall in front of web-facing applications.

*Note: This method is considered a best practice until June 30, 2008, after which it becomes a requirement.*

Full document at https://www.pcisecuritystandards.org/tech/download_the_pci_dss.htm

PCI-DSS is now managed by an industry consortium at www.pcisecuritystandards.org
Users shall be permitted to continue to use Compliant Products created or obtained prior to such termination.

6.2 Other than for Breach.

(a) In the event that Licensor believes that implementation of any Required Element(s) of the Specification infringes or may infringe the intellectual property rights (“IPR”) of an IPR Owner that is not willing to make such IPR Available under terms satisfactory to Licensor, then Licensor may (i) notify Licensee that it has amended the Specification, following which Licensee’s rights under this Agreement shall be limited to the Specification, as so amended, or (ii) terminate the License immediately.

(b) Licensee may immediately terminate the licenses granted in this Agreement upon written notice to Licensor.

7. Indemnification. Licensees shall indemnify, defend and hold harmless Licensor or its members, and the officers, directors, employees and agents of the same (each, an “Indemnified Party”) from all losses, costs, damages, claims and other expenses (including reasonable attorneys’ fees) arising out of any claim by any third party in connection with use by Licensee of the Material, including, without limitation, claims asserting that the Material or any portion thereof, infringes the patent, copyright, trade secret or other intellectual property anywhere in the world of such third party. Licensees shall indemnify, defend and hold harmless the Licensor and its members, and the officers, directors, employees and agents of the same (each, an “Indemnified Party”) from all losses, costs, damages, claims and other expenses (including reasonable attorneys’ fees) arising out of any claim by any third party in connection with use by Licensee of the Material, including, without limitation, claims asserting that the Material or any portion thereof, infringes the patent, copyright, trade secret or other intellectual property anywhere in the world of such third party.

8. Export Regulations. The technical data and technology inherent in the Material may be subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Licensees agree to comply strictly with all such regulations and acknowledge that it has the responsibility to obtain licenses to export, re-export, or import the Material and any Compliant Products.

9. Government Restrictions. Use, duplication or disclosure of the Material by the United States government is subject to the restrictions as set forth in the Rights in Technical Data and Computer Software Clauses in DFARS 252.227-7013(c)(1)(ii) and FAR 52.227-19(a) through (d) as applicable.

10. Miscellaneous.

10.1 Notices. All notices required under this Agreement shall be in writing, and shall be deemed effective five days from deposit in the mails. Notices and correspondence to (a) Licensee must be sent to the address shown above, and (b) to Licensee shall be sent to the address identified by Licensee in the form completed by Licensee below in the Agreement.

10.2 Governing Law. This Agreement shall be construed and interpreted under the internal laws of the United States and the State of Delaware, without giving effect to its principles of conflict of law.

10.3 Entire Agreement. This Agreement constitues the entire agreement and understanding between Licensor and Licensee regarding the subject matter contained herein. No modification or waiver of any provision of this Agreement shall be binding unless it is in writing and signed by both parties, and no waiver of any breach of this Agreement shall be deemed to be a waiver of any other or subsequent breach. If any provision of this Agreement is held by a court of competent jurisdiction to be invalid, illegal or unenforceable, such provision shall be omitted and the remaining terms shall remain in full force and effect. This Agreement supersedes any and all prior agreements between Licensor and Licensee regarding Licensee’s right to use the Material.
Update Notifications

Often users will obtain a product and never upgrade it. However, sometimes it is necessary for the product to be updated to protect against known security vulnerabilities.

How to identify if you are vulnerable

- Is there a method of notifying the owners/operators/system administrators of the application that there is a newer version available?

How to protect yourself

Preferably, the application should have the ability to “phone home” to check for newer versions and alert system administrators when new versions are available. If this is not possible, for example, in highly protected environments where “phone home” features are not allowed, another method should be offered to keep the administrators up to date.

Regularly check permissions

Applications are at the mercy of system administrators who are often fallible. Applications that rely upon certain resources being protected should take steps to ensure that these resources are not publicly exposed and have sufficient protection as per their risk to the application.

How to identify if you are vulnerable

- Does the application require certain files to be “safe” from public exposure? For example, many J2EE applications are reliant upon web.xml to be read only for the servlet container
Introducing the only tool in the world that really works effectively today......
News for people who run tools
A fool with a tool

....is still a fool
China!
China!
China!
People

Process

Technology
Fair and Balance

Automated tools aren’t totally “useless” today

(* but the marketing departments cards are marked)
What sort of tool do we want?

Testing framework / toolkit that combines
  Binary
  Run-time
  Code
  Pen
  AI (or human driven)
  Extensible
  Community driven rules
That’s all folks!