Misuse of Internal Code
A Case Study

Prepared by: Greg Elkins
Version: 1.4b
Date: April 2003

Executive Overview

One issue of information security that this instance highlights is the need to ensure the tenet of "need to know" is followed by internals. "Need to know" is the simple idea of ensuring that only individuals that require access to information to be able to complete their duties are allowed access, while others are blocked. Normally, the protection of data seems to be focused upon protecting the disclosure of sensitive data to externals. It should be noted that even though this is an important aspect of information security, the philosophy of "need to know" should apply to anyone that has access to the data, internally or externally to the organization (Liddy, Elizabeth. "Information Security and Sharing" May 2001. http://www.infotoday.com/online/OL2001/liddy5_01.html 20 Apr 2003), even at an application coding level. Although individuals in one department may have the perceived need and the technical expertise to be able to use the information, if they fail the "need to know" rule, their access to that information should not be allowed.

<u>Before</u>

The company provides customers with the ability to perform online research in multiple data areas. In order to allow customers to do this, the company provides two web applications. These applications, however, are relatively large and complex, due to the number of different methods that can be used to be able to search, display, and sort the retrieved data. The company had noticed that sales to certain markets were sluggish, with feedback from the marketplace stating that the web products were too complex and unwieldy.

The web applications were designed for the power information researcher, and not for the casual researcher. Via market research, it was found out that the markets would need to use the functionality that was provided by the existing web applications, but not all the features of the web application. This meant that either a new user interface (UI) to the existing website would need to be created, or some other solution would need to be created.

Due to the usage and exiting customer base that used the existing applications, the creation of a new UI would most likely alienate these existing users. The ability to provide two user interfaces had multiple technical concerns, most dealing with the ability to continue to provide the same high level of availability, performance and reliability that the customers depended upon. Making changes to the existing applications was deemed to risky for the business model and other methods of solving these issues were discussed.

The first solution offered would be to create a new web application specifically for these users. Unfortunately, since the markets would use the majority of the functionality of the existing web site, the new web application would basically be a duplication of the existing applications, including hardware and network connectivity. This proved to be cost prohibitive to the company and was discarded.

Therefore, the investigation of an Application Program Interface (API) was discussed and agreed upon. An API is a method to extend the functionality of some application so that it could be used by other applications. By providing an API, the existing functionality of an application would be used, and reused, without have to rewrite the code for every new application that wished to use the functionality. By providing an API into the existing applications, creation of new web applications would be easier, due to their being able to use the existing functionality of the current web application in a manner more suited to the customers that they were wanting to service.

The data that is being sent is primary XML data. XML (eXtensible Markup Language) is a method to mark-up data in order to provide more information about the data to the consumer. The provider of the XML document is able to define what the data is via the use of the XML elements. This is much like the way a web page designer will use HTML (Hypertext Markup Language) to tell a web browser how to display a page. Via the use of HTML elements, the browser is able to determine what text is which color, what links go where, and what images should be displayed. However, unlike HTML, whose elements are defined, XML allows the creator of the document to create and define their own elements used to describe the data. This allows great flexibility and extensibility that is not provided by HTML.

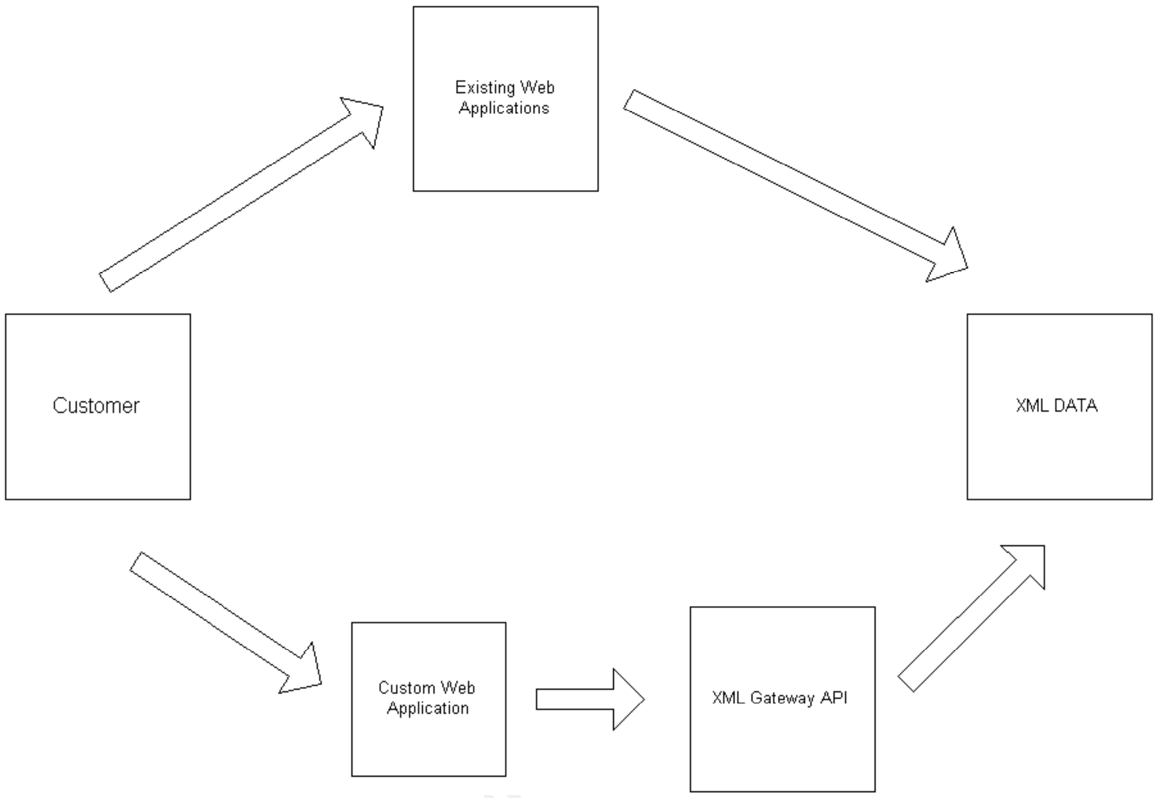
Due to the use of XML, the resulting API created was known internally as the XML Gateway. A bonus feature of the XML gateway was quickly discovered, as internal development groups would access the gateway to use the existing functionality within their own products. However, an issue did occur that was unforeseen. Due to the relative lack of technological expertise in the markets that the XML gateway was created for, multiple customers did not have the ability to use the gateway. This blocked those customers from being able to subscribe to the company's services.

With further research the company determined that the customers wanted a simple web application, tailored to their needs and only giving them the functionality that they wanted. The majority of customers did not want to create their own web applications, but would rather use something created and supported by the company. Part of this was due to the complexity of the API. The user manual for the gateway was over 400 pages and grew with each successive release.

Additionally, the usage of the XML Gateway was rather complex. Multiple parameters would need to be passed in the correct format and sequence for the transaction to succeed. Errors within the parameters could cause the transaction to fail, give incorrect results, or even bring down the gateway or effect the original application in a negative manner. To counter this, the developers of the XML Gateway made themselves available to train and also consult on any customer-side implementation of the Gateway. Unfortunately, even with the developers acting in a consulting role to customers, there were still multiple customers that were unwilling to use the Gateway.

At this point the company created a new development team which was to create the simple web applications for the customers. In order to be able to get the requirements from the customer, it was decided that the technical sales group would work with the customer, perform demonstrations, gather the requirements and the review the final application with the customer. The technical sales group was not trained on the use of the Gateway, due to that being an implementation role rather than a sales role.

The solution to be provided to customers would be the creation of a custom web application based upon the requirements gathered by the technical sales group. This web application would use the API within the XML gateway, and pass the search information to the Gateway for processing. The Gateway would process the request, and send the resulting information in XML back to the custom application, which would then render it. The custom page built to the customer could be a simple as a couple of forms built into a single page, to a multiple layer deep website, with added functionality so that the page could be linked to other sites. Client-side scripting, mainly JavaScript, was used for the creation of the API links, with little to no server-side scripting.



At this point the customers were delighted with the solution offered by the company. Sales of these customized web applications increased, along with the workload of the development team that created the applications. Unfortunately with the increased workload creating these custom applications, the ability for the development team to create demonstration applications for potential customers suffered.

With the turnaround time for demonstration pages becoming longer due to the higher priority of creating the sold custom web pages, the technical sales team found themselves needing to postpone sales calls, or rescheduling them at a later date. The technical sales team took it upon themselves to start creating their own demonstration pages via the reverse engineering of the developer-created custom web pages.

Since the custom web page was nothing more that HTML and JavaScript (perhaps a cascading style sheet based upon what the customer wanted), there was no problems with going to an existing page in a browser, selecting view source, and then cutting and pasting the JavaScript into their new page. They would then host these pages on any server that they could, internally or external to the company.

At this point, the company incurred multiple risks. Individuals who were not trained in the use of the Gateway were creating web applications that interfaced with the Gateway. Due to the complexity of the Gateway, errors within the requests submitted to the Gateway could cause negative impacts on not only the Gateway and the custom applications it services, but also the existing web applications.

Additional risk was incurred via development efforts done by the technical sales group on the demonstration pages not going through existing company policy and process. By not following the current processes and adhering to the existing policies, the company was opened up to the threats of not only impacts to existing products, but also impacts to the systems that were hosting the rogue web applications. The rogue applications were being hosted on servers that did not have the processing power or memory to be able to handle the additional load of the rogue application.

More risk was incurred by the use of non-company servers to host some of the demonstration applications. Not only was no support of these servers guaranteed, but it was found out that some customers continued to use the demonstration pages hosted externally to the company even after the final custom web application was finished. The external servers did not have any form of Service Level Agreement that detailed that availability and support of the servers. Additionally, since the servers were not underneath the company's control, there was a threat that someone could 'steal' the demonstration application that contained proprietary code.

During

The company's department that monitored the status of the applications found the technical sales group's activities. At times, traffic was coming to the Gateway from addresses on the Internet that did not match either the company's address range, or match any of the customers. At this point, information security was contacted.

The company currently has a dedicated information security team, and has policies covering information classification, information security, network security and application security. These polices are published on the company's intranet and copies are provided to employees during their information security training they receive when they first start at the company.

Information Security first reviewed the situation to determine whether existing policy covered this scenario and found the following violations:

- Hosting of web applications on third-party servers
- Non-development group did development work
- Applications that did not go through the product development process were exposed to customers

These violations exposed the company to the following risks:

- Risk to company's reputation for hosting applications in environments that were not underneath the company's control.
- Risk to company's revenue due to failure of an application in a third-party environment would cause the application to be inaccessible
- Risk to company's products with the threat of non-conforming requests being sent from non-development created applications being sent to the XML Gateway.
- Risk to company's reputation and revenue for having a product that was not fully certified to company standards due to it not going through the company's product development process.

Since these were covered by policy, it was determined to first find out why this issue occurred.

Meetings were held with the technical sales group to review the existing policies. Although every employee at the company has initial training on the existing information security polices, it was found out that the current ongoing awareness training was rather weak, and needed to be tailored for each group that it was presented to. For instance, the current awareness program tended to focus on the information classification policy and the procedures to protect data for the non-development groups where the development groups got training mainly on the policies that governed the applications that they built.

With the technical sales group's lack of knowledge of the application security policy along with the network security policy, they believed that their actions were within the guidelines of the company. Their actions did fall under the information classification policy and information security policy, since they did have controls around the access of the demonstration applications they created.

At this point, Information Security reviewed the application and network security polices with the technical sales group and their management to educate them on the policies. Reviewing the network security policy detailed that in order for an application to be served to the customer, certain hardware and network specification needed to be met. Additionally, 24 by 7 support would need to be provided to ensure that the application was available to the customers at all times. After the review of the network security policy, the technical sales group was aware that the only allowable location for hosting applications would be within the company's own hosting center.

Review of the application security policy raised some eyebrows in the technical sales group. The current application security policy states that there is to be three environments that are to be used for developing applications, first of which is the Development environment. The Development environment is used for the initial development of applications and is not accessible from the Internet. There are no controls on the code, developers are able to work the code as they see fit.

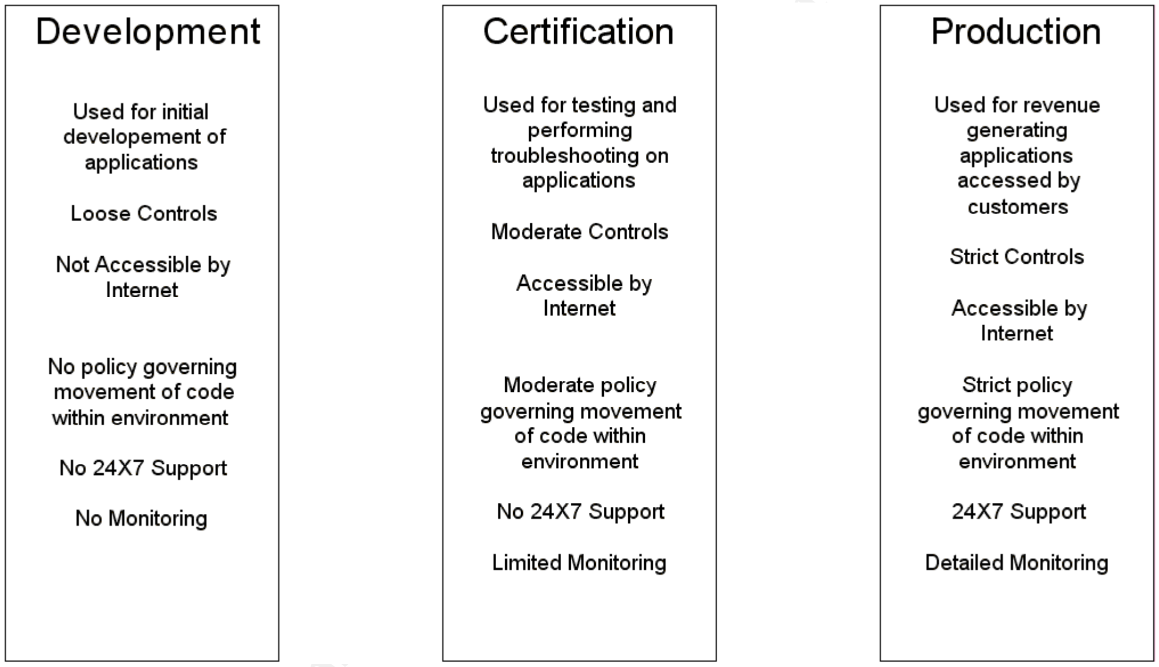
The second environment is Certification. The Certification environment is used mainly for testing and usability of the code created in the Development environment. There are higher controls upon the code within the Certification environment, along with high levels of service on the infrastructure that makes up the environment. Code is rolled into the environment via a defined process by the developers. However, unlike the Development environment, the process ensures that there is documentation on what code was moved and who is responsible for it.

Unlike the Development environment, the Certification Environment is accessible from the Internet. This is due to the need to provide usability testing for some of the applications. There is a process that is in place to grant access to the applications for this testing. Additionally, the certification environment is used to test known issues with customers to ensure that they are resolved prior to release to the Production Environment.

The Production Environment is where the applications that are accessed by paying customers are located. A highly secure, Internet accessible environment, it is monitored 24 hours a day to ensure that availability, reliability and performance are maintained. Issues within this environment are treated at the highest levels by the support organizations to ensure that there is limited impact to customers.

Access to the applications within the Production environment is strictly controlled. Prior to any code being placed in the production environment, exhaustive testing within the Certification environment must be performed, and the code must show that there are no issues that would affect either its own application, or any applications that also live in the Production environment. The process for moving code into production requires not only documentation of what code is being moved and who is requesting it, but also documentation regarding the certification testing, and a managerial approval for the movement of the code.

Code moves are scheduled and are done by a team of individuals who are responsible for the Production environment. Any movement of code into Production also requires steps to be able to reverse the move in case any issues arise once the new code is in the Production environment.


Development
Used for initial developement of applications
Loose Controls
Not Accessible by Internet
No policy governing movement of code within environment
No 24X7 Support
No Monitoring
Certification
Used for testing and performing troubleshooting on applications
Moderate Controls
Accessible by Internet
Moderate policy governing movement of code within environment
No 24X7 Support
Limited Monitoring
Production
Used for revenue generating applications accessed by customers
Strict Controls
Accessible by Internet
Strict policy governing movement of code within environment
24X7 Support
Detailed Monitoring

The demonstration applications created by the development group were located in the Certification environment, whereas the final custom product would follow the existing process and be placed in the Production environment. This would allow the development team to not only be able to work with the customer to ensure that the application they were creating was sufficient to fulfill the customer's needs, but also have the testing done that is required to move the custom application into production.

However, the technical sales group's hosting of the demonstration application outside of these environments caused the company to incur risks due to the threat of not allowing full testing to take place under controlled conditions. Additionally, since some of the demonstration sites were being used as actual products by some customers, the ability to provide a high level of support was lost, thus causing a risk to not only the ability for the company to be able to charge for services, but also the reputation of the company providing quality products.

An additional item within the application security policy is that developers were to build applications to pre-defined guidelines, based upon the recommendations of the Open Web Application Security Program (Curphey, Mark. Et al. "A Guide to Building Secure Web Applications" The Open Web Application Security Project. 11 Sept 2002. http://aleron.dl.sourceforge.net/sourceforge/owasp/OWASPGuideV1.1.1.pdf). Most of the guidelines were already being followed in development: bounds checking, sanitizing input, and the like. However the Company's application security policy differed from the OWASP guide in one way.

Under the section of reuse, the OWASP guidelines states: "Using and reusing trusted components makes sense both from a resource stance and from a security stance. When someone else has proven they got it right, take advantage of it." The Company's stance is not to forbid the reuse of components, but to have a peer-review of the component to be reused to ensure that both security and functionality issues will not exist. However, by reverse-engineering existing pages, the code could not be considered trusted, especially since the technical sales group was not having a code review being done.

By not doing a code review, not only was the technical sales group not following policy, but increasing risk due to the ability for code that may not be specifically designed to do what it is tasked to do attempting to interface with the XML Gateway. With this review, there was an interesting point made. Since the web application was mostly JavaScript embedded in the page, what would keep the customer from doing exactly what the technical sales group did?

JavaScript is a programming language that can either be run by the web server, or by the browser. For the script to be run by the browser, it is usually placed either in the web page itself, or in a separate file that is downloaded and referenced by the page being processed by the browser. In order to view the JavaScript, the end-user would simply use the View Source option that is within the majority of browsers on the market today. At that point, it would be quite easy to be able to determine how the code is written and allow the end-user to modify or create a web application that would use the Gateway.

This presented a challenge, either modify the API to ensure that the risk of customer crafted API calls did not negatively impact the company's applications, or ensure that the requests going to the XML gateway were generated by the application created by the company's development group. Although the risk of damage to the company's reputation was limited due to the customer creating the web application, the risk of an improperly crafted request causing issues with the Gateway and perhaps affecting the company's other applications still existed.

Modifying the API to any great extent to minimize the risks of improperly crafted requests was not a feasible option, due to the complexity of the task and resources available. This decision was also impacted by management's decision to retire the gateway application within two years, and move individuals to a new platform currently under development.

Not offering the functionality that caused the risks was also deemed unacceptable. Multiple customers had signed contracts making the company obligated to offer the functionality for the life of the contract (in some cases, these contracts ran for multiple years). The solution was to modify the application in such a way that it would only accept requests generated from authorized web pages.

Three methods were discussed; the first was IP authentication. Using the source IP address of the requestor to validate that the user was coming from a known site would reduce some of the risk, but not eliminate it. However, the in-depth discussion dismissed this as causing more issues than resolving.

The main idea of the use of IP authentication is that only certain source IP addresses would be able to access the correct page that would have the links. Unfortunately, since the traffic is via HTTP, this link is a normal hyperlink, causing the client to make a new call to the application, thus still not allowing the application to determine whether or not the request came from the authorized page.

An additional risk that the company was unwilling to accept would be that the inclusion of an IP authentication scheme could be used as a method of launching a Denial of Service attack. A Denial of Service attack (DOS) is where a tremendous amount of invalid traffic is directed to an application so that the application spends all its time dealing with the unauthorized traffic and cannot process authorized traffic.

The XML gateway would be vulnerable to a DOS attack due to the protocols being used are HTTP and HTTPS. Neither protocol has the mechanisms to maintain state, thus causing each transaction needing to be authenticated. This would place a rather sizable load on the database that would be handling the authentication and authorization information. A malevolent user would easily be able negatively impact the performance experienced by users by simply sending multiple requests from unauthorized IP addresses.

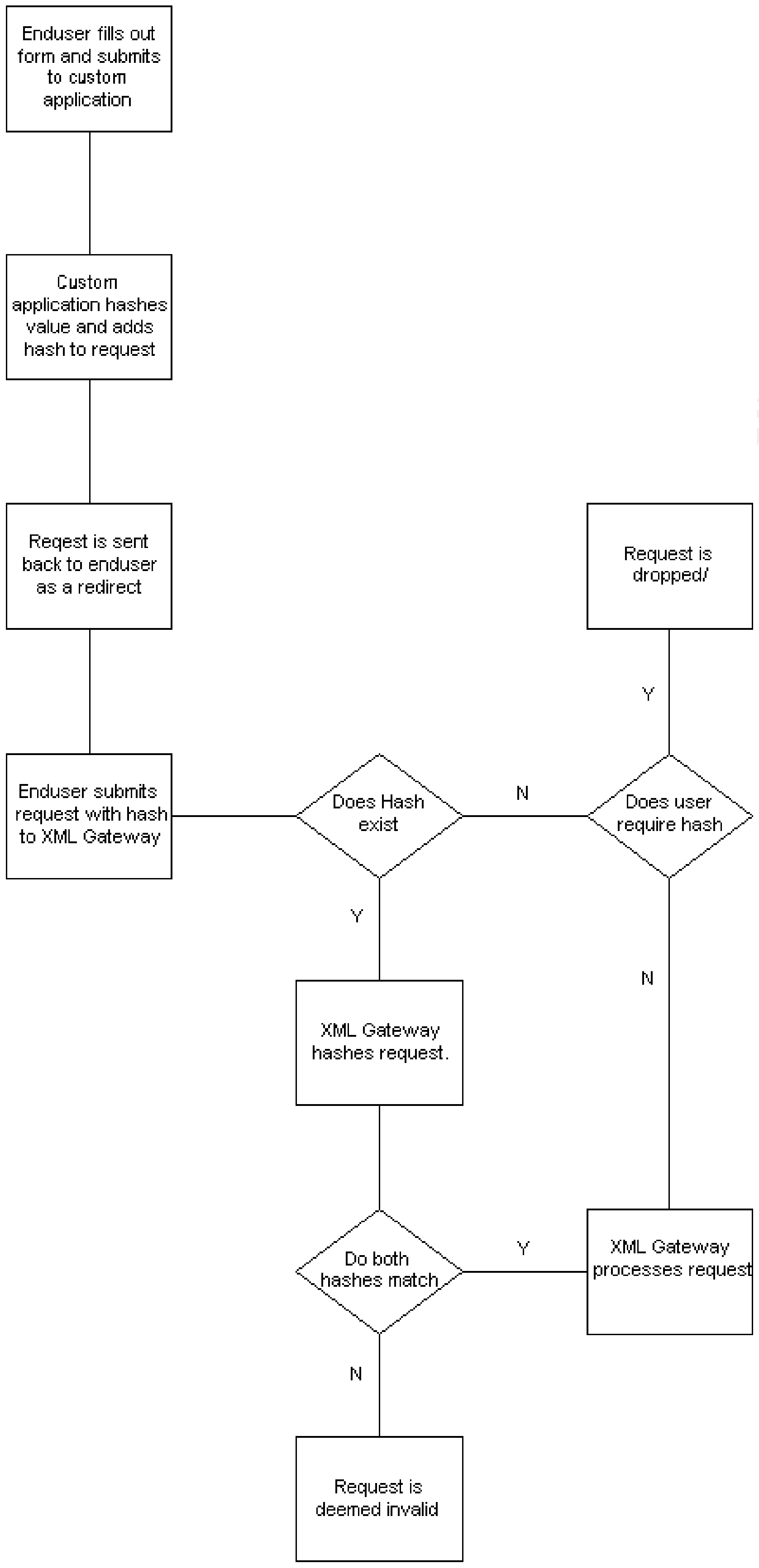
The second resolution examined was an attempt at using the HTTP referrer header. By ensuring that the web page that sent the client to the API was a valid web page, it could be safely assumed that the page was an authorized page. However, per the RFC that covers the implementation of the HTTP protocol (Fielding, R. et al. "Hypertext Transfer Protocol – HTTP 1.1" Network Working Group Request for Comments 2616. June 1999. http://www.faqs.org/ftp/rfc/rfc2616.txt) the use of the referrer field is discouraged due to the header allowing individuals to read patterns that can be studied and allow reverse links to be crated. Additionally, multiple firewalls and network security devices can be set up to ensure that header is stripped.

The final method discussed, and finally accepted was the use of a security token, that would be created and passed along to the client. Any modification to this security token or to the request URL created by the custom web page would cause the API to believe that the request was invalid, and therefore not process this request.

This discussion morphed into a discussion of the use of a digest of the link as a digital signature. A digest is a method of being able to determine that a message has not been altered via use of a hash. Much like how a fingerprint is unique for an individual, a digest would be unique for a message. The hash is value calculated by sending a message through an algorhythm so that a value is created. The issue is to find an algorhythm that it is rather difficult to determine how the hash value was calculated.

The process of authenticating the validity of the message as applied to this situation would be:

- End user enters their request parameters into a form and presses submit.
- The custom web application takes the request, and runs it through a hashing algorhythm to calculate the message hash.
- The customer web application appends the hash to the message and forwards the request to the XML Gateway.
- The XML gateway takes the message and separates the hash from the request. If no hash value is found, the message is discarded.
- The Gateway calculates the hash of the message and compares the hash it generated to the hash that the message contains.
- If the hashes are the same, the gateway processes the request. If the hashes are different, the gateway will discard the request and generate an error message.


Enduser fills out form and submits to custom application
Custom application hashes value and adds hash to request
Reqest is sent back to enduser as a redirect
Enduser submits request with hash to XML Gateway
Does Hash exist
N
Does user require hash
Y
Request is dropped/
Y
XML Gateway hashes request.
N
Do both hashes match
Y
XML Gateway processes request
N
Request is deemed invalid

The use of MD5 as the hash mechanism was selected due to the use of the algorithm in other products that the company currently has and would not need to go through the approval process. The use of the digest could be seen as a form of a digital signature that allows the integrity of the data signed to be assured. (whatis.com. "Digital signature" SearchSecurity.com Definitions. 16 Apr 2001. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci211953,00.html)

Another issue was to keep the end from being able to determine how to create the digest. Although MD5 has methods so that the recipient of a digest cannot determine how it was calculated, (Advosys Consulting. "Preventing HTML Form Tampering" 08 Mar 2003. http://www.advosys.ca/papers/form-tampering.html) management wanted to ensure that the customer did not know what method of digesting was being used. If the end-user was able to determine that MD5 was being used, and the key that was being used with it, then the digests created by the end-user would be valid, thus nullifying the controls.

At this time, it made sense to create a server-side script that would be used to create the digest. This server-side script was placed in a folder within the production web server (after full testing via the process described earlier), with controls that only allowed the page itself and development the ability to access the script. This not only kept the risk of third parties attempting to write calls to the XML gateway, but also ensured that the company's employees were unable to do the same.

This access limitation initially created some concern for the support organization, which required access to the source code of the custom web pages for support purposes and testing. However, through working with development, they were able to determine that this would not have a major impact on the support duties, and any tasks that required the ability to access the hashing script would be escalated to the development group.

By making these modifications, the API required that all existing custom web applications accessing the Gateway to be updated to ensure that they were not broken when the modifications went into affect. A rather sizable effort was done to ensure that all of the existing custom web pages were updates, and exhaustively tested by internals prior to releasing the new API and web applications to customers. By doing the exhaustive testing, multiple issues were noted, mostly coding mistakes, and did not impact customers at all.

Finally, issues were brought up regarding the customers that were technically savvy enough to use the XML Gateway on their own. With the current plan, their requests would not have the correct hash value and would be rejected. This was resolved by making the digesting feature as being unneeded on their IDs that accessed the Gateway.

After

As stated earlier, not all security risks come from externals or internals with malevolent goals, but can be caused by internals who attempt to do their jobs. Although a company may have a clearly defined information security policy, publish it and document that employees agree to be bound by the policy, it does not mean that individuals understand the policy or retain the information for long periods of time

This incident caused Information Security to reassess their training program and ensure that not only did initial training cover the necessary policies and procedures, but the ongoing training would focus on how the Information Security polices affected individual departments. Modifications to how the ongoing training was delivered also occurred. Rather than have web-based training modules, the departments needing training may request an individual from the Information Security department give the training and be available to answer questions and solicit feedback.

It was also noted that the information security policy would have to be revisited and updated due to the issues that this scenario disclosed. Prior to the XML Gateway, the company had strict controls over the business logic used by applications. By creating the API for the XML Gateway, the company created a method that could be used by externals to provide their own business logic and create their own applications. Modifications to the application security policy were added to ensure that the validity, integrity and confidentiality of the requests made through the XML Gateway were maintained.

With the further education of the technical sales group, a relationship between that group and information security has grown to the point where the sales group sees the information security group as a resource to be used to help win new business for the company. Additionally, feedback provided from the technical sales group resulted in a customer version of the existing information security polices to be created to assist in the sales project.

For the application, overall risk to the company's reputation and revenue were reduced substantially via the implementation of the security policies. With the technical sales group moving out of the development business along with ensuring that hosting of both the demonstration and production versions of the custom web applications at the company's hosting site helped reduce the overall risk that the company previously had. Additionally, the issue of ensuring that the requests to the XML Gateway were coming from applications created internally, a currently undefined risk was reduced.

Overall, this scenario reinforced the idea that information security not only needs to worry about external threats, but also internal threats. Although it has been noted that the majority of malicious activity that causes damages to information technology occurs from what appears to be internal users (Thompson, Kerry. "The Travesty of Internal Security" 24 April 2002. http://www.crypt.gen.nz/papers/internal_security.html), it seems that focus tends to be on those users that wish to cause some form of damage to the company. However, as this paper indicates, even those individuals who wish to help the company may cause additional risk by not following policy and process and must also be integrated into any information security program.

References

Advosys Consulting. “Preventing HTML Form Tampering” 08 Mar 2003. http://www.advosys.ca/papers/form-tampering.html

Curphey, Mark. Et al. “A Guide to Building Secure Web Applications” The Open Web Application Security Project. 11 Sept 2002. http://aleron.dl.sourceforge.net/sourceforge/owasp/OWASPGuideV1.1.1.pdf

Fielding, R. et al. “Hypertext Transfer Protocol – HTTP 1.1” Network Working Group Request for Comments 2616. June 1999. http://www.faqs.org/ftp/rfc/rfc2616.txt

Liddy, Elizabeth. “Information Security and Sharing” May 2001. http://www.infotoday.com/online/OL2001/liddy5_01.html

Thompson, Kerry. “The Travesty of Internal Security” 24 April 2002. http://www.crypt.gen.nz/papers/internal_security.html

whatis.com. “Digital signature” SearchSecurity.com Definitions. 16 Apr 2001. http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci211953,00.html