

Best Practices for a Secure “Forgot Password” Feature

By Dave Ferguson, Principal Consultant • CISSP, CSSLP

Introduction

As an application security consultant, I’ve had the opportunity to examine a wide variety of web applications. If I were to count the number of applications I’ve assessed, there would probably be the same number of different techniques for handling forgotten passwords. It seems everyone has their own idea of how it should be done. There is no industry standard, and developers tend to bolt on the functionality as an afterthought. The result is that users could be forced to jump through myriad hoops involving emails, special URLs, temporary passwords, personal security questions, and so on. In some applications you can recover your existing password. In others you have to reset it to a new value.

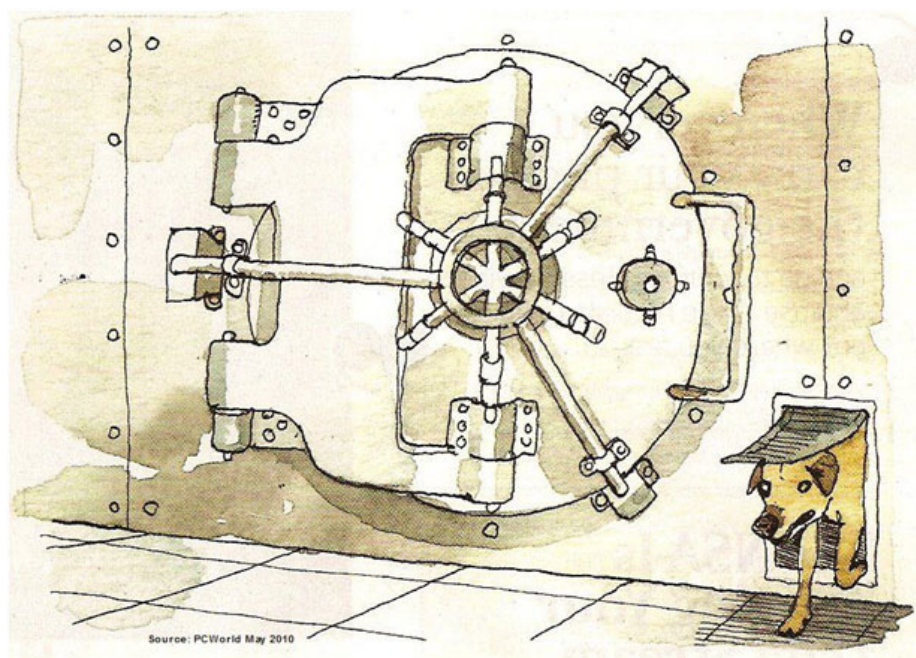
Which techniques are best from a security standpoint? In this paper, I will describe how to implement a secure forgot password feature. The information presented has been extracted from my experience in assessing the security posture of web applications.

Note: The recommendations presented in this paper are most appropriate for organizations that have a business relationship with users. Web applications that target the general public (social networking, free email sites, etc.) are fundamentally different and some concepts presented may not be feasible in those situations.

A Tricky Proposition

Web applications typically authenticate users via username and password inputs. The authentication process may receive plenty of attention during application design and development, and much has been written about how to secure this process:

- Enforce strong user passwords.
- Disable the account after a certain number of failed login attempts.
- Display generic error messages.
- Store passwords in the database as salted hash values.



However, all of these carefully thought-out security measures can go out the window when a user has forgotten his or her password. As a user, your expectation is that the application will assist you when you forget your password. But there's something fundamentally insecure about this. Essentially, the application is providing a second, entirely separate procedure to authenticate you.

It is this secondary method of authentication that often gets short shrift. Security weaknesses creep in. Let's first explore some existing techniques for resetting or retrieving a lost password.

Existing Techniques

The following techniques are examples of how web applications help users when they've forgotten their password.

- A - Email the current password
- B - Email a temporary password
- C - Email instructions on how to reset the password
- D - Display the current password after answering questions
- E - Provide a password hint or password reminder

There are other techniques, but the five above are quite common. Let's review these in more detail. For the sake of this exercise, let's say that a user named "Joe" can't remember his password.

Weak Technique A – Email the current password

With Technique A, Joe is asked to enter either his username or email address. The application looks up the email address for the username provided (or verifies that the email address is valid) and proceeds to send an email to Joe with his current password. This is one of the simplest and most common techniques used by web applications, but it is also one of the least secure. The fundamental weakness here is that email is not a secure form of communication. Sensitive data is transmitted unencrypted over the Internet, meaning a hacker who is sniffing network traffic could steal Joe's password. It could also be stolen if another person could view Joe's emails or had broken into his email account.

Unfortunately, applications can aggravate the situation by including Joe's login ID (username) in the same email. Hackers like this, because now they have Joe's full login credentials not just his password. Of course, if an application is using email address as the login ID, there's no way to avoid this ugly outcome.

Another concern with this technique is storage of Joe's password. It is not stored as a one-way hash value, which is a best practice to keep passwords safe. Current passwords are retrievable, so it is obvious to the world that they are being stored in the site's backend database either unencrypted or encrypted with a reversible, symmetric algorithm.

An example of Technique A from the real world is presented below (selected portions removed to protect the site's identity). In this case, the application has made a poor decision to send both the username (called "handle") and the password in the same email.

Figure A1 - You're asked to enter your email address Figure A2 – Message indicates that an email has been sent to you

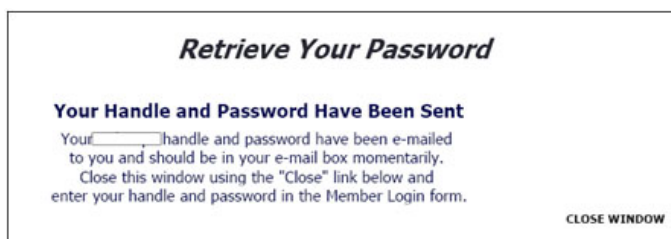


Retrieve Your Password

Forgot Your Password?

Just enter your e-mail address below
and we will e-mail your handle and password immediately.

[CLOSE WINDOW](#)



Retrieve Your Password

Your Handle and Password Have Been Sent

Your handle and password have been e-mailed
to you and should be in your e-mail box momentarily.
Close this window using the "Close" link below and
enter your handle and password in the Member Login form.

[CLOSE WINDOW](#)

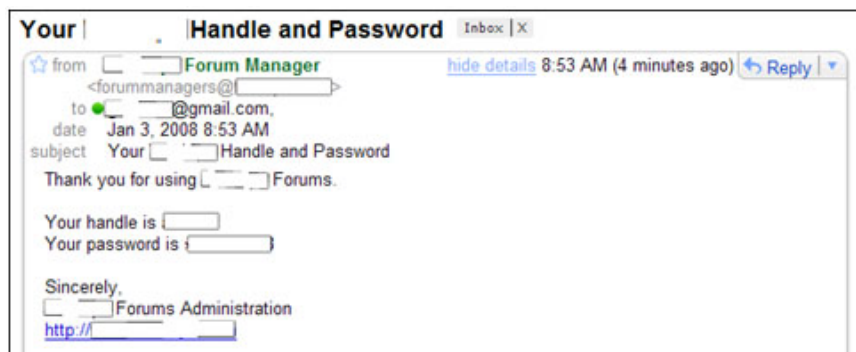


Figure A3 – Email contains your current password (and username too in this case)

Weak Technique B - Email a temporary password

Technique B is often used when an application is correctly storing passwords as one-way hash values. This is a good practice, but things go downhill from there. With this technique, Joe is asked to enter either his username or email address. The application proceeds to reset his password to a temporary value and immediately sends Joe an email containing the temporary password. The temporary password may be recognized only for a short time. Sometimes there is no time limit. Regardless, Joe is forced to change his password after logging in with the temporary password.

These steps make the whole process seem secure, but is it? Not really. The problem again is that passwords are sent via email. A hacker sniffing network traffic could compromise Joe's account. Creating a temporary password and forcing Joe to change it upon login are not effective countermeasures against network sniffing or a person who has hacked Joe's email account.

An example of Technique B from the real world is presented below.

Figure B1 – User asked to enter email address

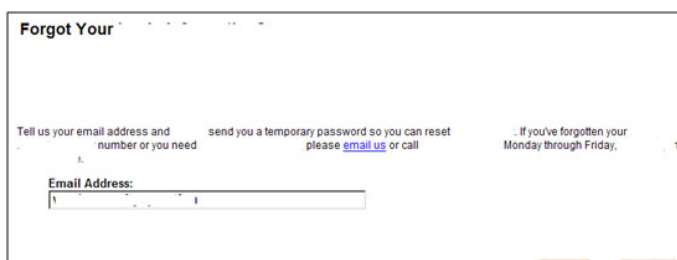


Figure B3 – The email provides a temporary password

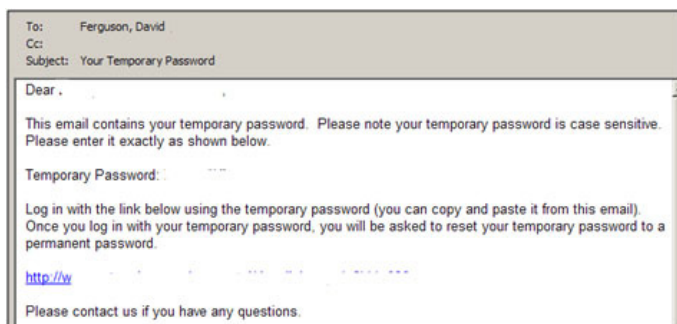


Figure B2 – Message indicates that an email has been sent

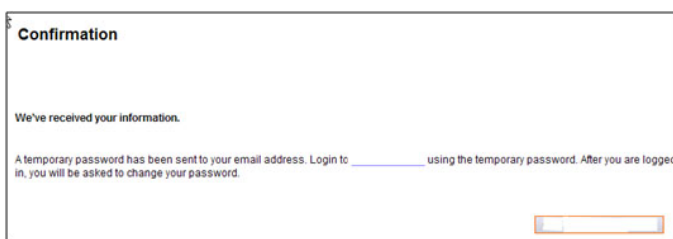


Figure B4 – After logging in with the temporary password, you immediately must change your password



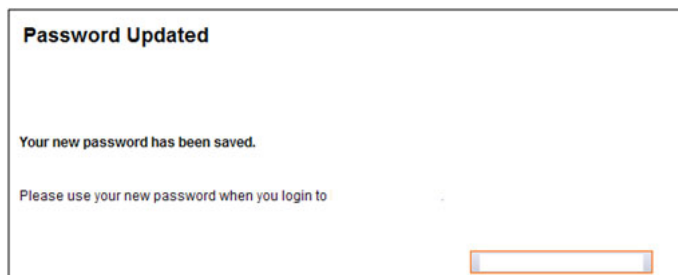


Figure B5 – Message indicates that the new password is in effect

Weak Technique C – Email instructions on how to reset the password

Technique C is similar to B in that Joe is asked to enter either his username or email address. The application then emails “instructions” on what to do next. A bit of obfuscation is at work because the application doesn’t at first explain what the next steps are. Indeed, the instructions are often different from one application to another. A temporary password may or may not be involved. Often, the email simply contains a special URL and Joe is asked to click on the link. The returned web page then allows Joe to reset his password. Unfortunately, Joe’s username may be shown on the page, or it may have been in the email itself. This technique is really no more secure than any of the other methods because email is again the decisively weak link.

A real-world example of Technique C is presented below. Note that the application displays the username on the last step. This is not necessary and is quite helpful to an attacker who has been able to steal the URL contained in the email.

Figure C1 – You’re asked to enter your email address Figure C2 – You receive an email with a URL containing a cryptic parameter

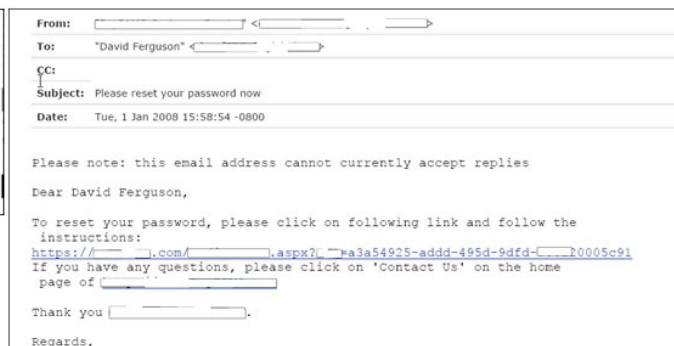
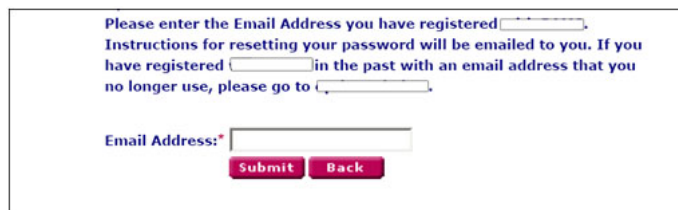


Figure C3 - Clicking on the URL opens a page that displays your username and allows you to reset your password

Weak Technique D – Display the current password after answering questions

Technique D is substantially different than the others because it does not rely on email. This is a good practice. In this technique, Joe is asked to enter several pieces of information about himself. The information ostensibly consists of shared secrets between the user and the application. Examples of these secrets are social security number, email address, zip code, account number, and date of birth. Personal security questions might be employed as well.

The security flaw with this technique is that a user's password should never be displayed on a web page. The page could be cached by the browser to the local hard drive where it could be viewed by unauthorized parties. Or, the password could be stolen by a shoulder-surfing co-worker. It also the fact that the site does not store passwords in the database as one-way hash values as it should.

Three examples of Technique D, taken from the real world, are presented below.

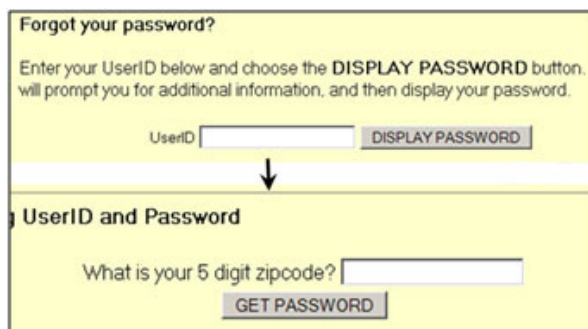
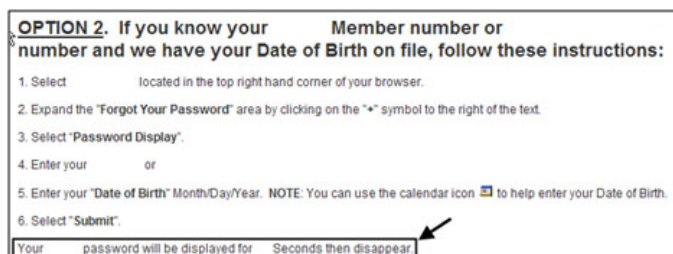


Figure D1 – An example where you enter username, then answer a personal question

Figure D2 – An example where you enter two pieces of information, after which your password is displayed



Figure D3 – An example where you can view your password, but only for a certain number of seconds



Weak Technique E – Provide a password hint or password reminder

Technique E was very common in the early days of dynamic web applications, but seems to have lost favor in recent years. Like Technique D, it does not rely on email, and passwords are not displayed on the screen (theoretically anyway). In this case, Joe has previously entered a word or phrase to help him remember his password.

The real trouble with this technique is that it implicitly assumes that a word or number meaningful to the user was chosen as the password and that the user just needs a bit of information to help jog his memory. Meaningful words and numbers are typically not strong passwords, so applications of this type tend to encourage weak passwords. Furthermore, most users are not security experts. The particular hint that Joe chooses could divulge too much information, making a hacker's job very easy. Joe might even decide to use his actual password as his password hint!

A real-world example of Technique E is presented below.




Figure E1 – You're asked to enter your username to see your password hint



Figure E2 – Your password hint is displayed (in this case the password is probably the make and/or model of a car)

A Secure Forgot Password Feature

Enough talk about weak techniques. What are the elements of a secure forgot password feature? The bottom line is that a user should be allowed to reset his own password by following a series of steps. Personal security questions should be used. The web application doesn't need to use email, display passwords, or set any temporary passwords.

Below is a step-by-step (or page-by-page if you will) recommendation for a secure forgot password feature.

Step-by-Step Password Reset

Step 1 – Get hard data

The first page of a secure forgot password feature asks the user for multiple pieces of hard data. A single HTML form should be used for all of the inputs.

A minimum of three inputs is recommended, but the more you require, the more secure it will be. One of the inputs, preferably listed first, should be the username. Others can be selected depending on the nature of the data available to the application. Examples include:

- email address
- last name
- date of birth
- account number
- customer number
- last 4 digits of social security number
- zip code for address on file
- street number for address on file

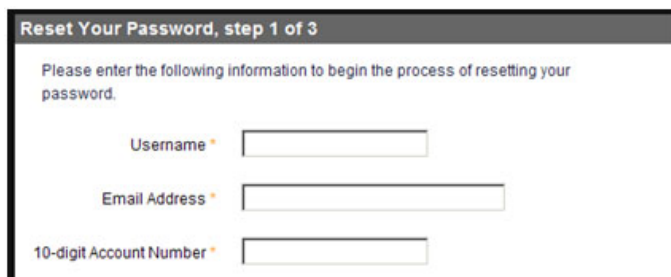
Step 2 – Ask personal security questions

After the form on Page 1 is submitted, the application verifies that each piece of data is correct for the given username. If anything is incorrect, or if the username is not recognized, the second page displays a generic error message such as "Sorry, invalid data". If all submitted data is correct, Page 2 should display at least two of the user's pre-established personal security questions, along with input fields for the answers. It's important that the answer fields are part of a single HTML form.

Do not provide a drop-down list for the user to select the questions he wants to answer. Avoid sending the username as a parameter (hidden or otherwise) when the form on this page is submitted. The username should be stored in the server-side session where it can be retrieved as needed.

Step 3 – Allow user to reset password

When the answers to the security questions are submitted on Page 2, the application verifies that the answers are correct. If they aren't, the user is returned to the same page where he has another chance to answer. If the submitted answers are correct, Page 3 should (finally) allow the user to reset his password. Display a simple form with one input field for the new password and another field to confirm the new password. Make sure to enforce all password complexity requirements that exist in other areas of the application. As before, avoid sending the username as a parameter when the form is submitted.



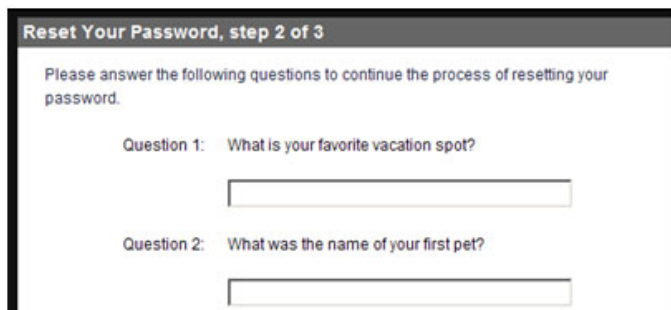
Reset Your Password, step 1 of 3

Please enter the following information to begin the process of resetting your password.

Username *

Email Address *

10-digit Account Number *




Reset Your Password, step 2 of 3

Please answer the following questions to continue the process of resetting your password.

Question 1: What is your favorite vacation spot?

Question 2: What was the name of your first pet?



Reset Your Password, step 3 of 3

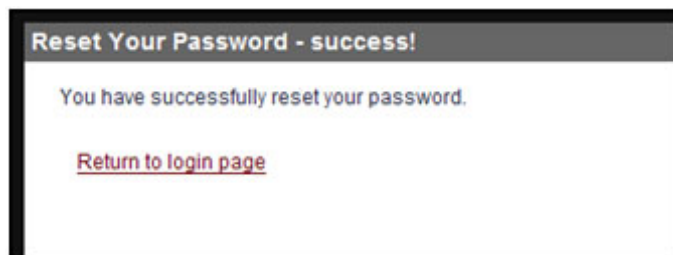
Please reset your password below.

New Password

Re-enter Your New Password

Step 4 – Display success message

Display a “success” message. Explain that the user has successfully reset his password. Provide a button or link to return to the normal login page of the application. An example is shown below.



For Extra Protection

You can raise the security bar even further. After step 2, email the user a randomly-generated code having 8 or more characters. Require input of the correct code on step 3, along with the password inputs. This introduces an “out of band” communication channel and would be extremely tough for a hacker to overcome. If the bad guy has somehow managed to successfully get past steps 1 and 2, he is unlikely to have compromised the user’s email account. The downside with this approach is that it creates a problem for legitimate users who have an obsolete or invalid email address stored within the system.

DO’s and DON’Ts

Here are some important DO’s and DON’Ts to keep in mind when designing a secure forgot password feature.

DON’T allow users to bypass any steps

Hackers use a technique called “forced browsing” to bypass application logic. Your application should ensure the user has completed every step in the password reset process. For example, use a Boolean flag in the server-side session to know when a user has correctly answered the personal security questions. Display an error if a user attempts to skip that page by navigating directly to the final password reset page.

DO establish personal security questions with each user

As part of each user’s profile, establish at least three personal security question and answer sets. In general, you should provide five unique questions for every one question that is needed. For example, provide a list of 15 questions to choose from when three questions are required. Supply a pre-defined list of questions. I don’t recommend allowing users to create their own questions. Doing so opens the possibility for weak questions such as “What is your favorite color?”. The additional inputs also serve to increase the attack surface of the application. The additional inputs must be thoroughly validated and encoded or else cross-site scripting or some other type of attack may be possible.

DON’T permit weak security questions

Assess the strength of the security questions you make available to users. Weak questions could lead to a compromised account, especially when valid usernames are disclosed. Case in point: A certain application I looked at recently had a problem with disclosure of valid usernames. It also asked two personal security questions. I found several users who were asked the following questions: *What month is your wedding anniversary?* *In what state were you born?*

It wasn’t difficult to see that only 600 combinations of answers are possible (assuming the users answered accurately). Hacking into these particular accounts was fairly easy, because there was no limit on the number of attempts. Needless to say, I recommended that our client remove these two questions from their list and replace them with stronger ones.

DON’T disclose valid usernames

Valid usernames should be a closely guarded secret. Often, I find that the primary authentication method properly displays a generic “invalid login” message, but that the forgot password feature does not. For example, the initial page of a forgot password feature should not have just a single input for username or email address. With only one input

on the form, the next page inevitably tells a user whether the username entered was valid or not. This type of account enumeration is dangerous. Remote hackers could run a brute-force or dictionary attack to pry their way into accounts. Or, if your application locks accounts after a certain number of failed login attempts, a denial-of-service attack could be performed against users.

DO implement account lockout

For an extra measure of protection, lock the user's account after a certain number of failed attempts to complete a step. Lockout simply means the application refuses all unauthenticated activity on the account. I suggest allowing between 5 and 10 attempts before triggering lockout (the primary login page should allow fewer). It's usually best to automatically unlock accounts after a certain time period such as one hour. Alternatively, you can require users to call in to request an unlock, but this is obviously more expensive and may require planning and coordination with other departments in your organization.

DON'T rely on email

Your forgot password feature should not be dependent on email, because it is an inherently insecure form of communication. You should take the mindset that emails sent by your application will be viewed by unauthorized parties. Furthermore, you will avoid the dilemma that occurs when a user's email address is no longer valid. Note that email is fine to use as a secondary channel for delivery of a random code (see "For Extra Protection" above). Email can also be used as explained in the next item.

DO send an email notification indicating the password was changed

After a successful password reset, the application should automatically send an email notification to the email address on record. The email shouldn't contain any password – just simply state that the password was changed. Doing so may alert a victim that his or her account was compromised. Otherwise, it's just a harmless, secondary confirmation.

DO store answers to personal security questions as one-way hashes with a salt

Similar to how passwords are treated, answers to personal security questions should be combined with a salt and stored in the database as one-way hash. Use a strong, industry-standard algorithm, such as SHA-256. For the salt, consider using the username or email address. To test incoming answers for validity, combine the answer with the salt, compute the hash, and then compare to the value stored in the database.

DON'T allow the HTTP GET method

Your application should accept only POST requests when sensitive data is sent by the client. Parameters sent with a GET request become part of the URL, while those sent with POST are sent in the HTTP request body. Requests for your forgot password feature will include sensitive data, so using GET causes sensitive data to appear in URLs, which are recorded in the browser history as well as web server logs.

DON'T allow username in the final HTTP request

When the application finally receives the HTTP request for resetting a user's password, it should be fully aware of which user's password is being reset. The username should be stored in the server-side session. Don't accept or recognize a username input on this final request. Attackers might tamper with the value.

ABOUT FISHNET SECURITY

We Focus on the Threat so You can Focus on the Opportunity.

Committed to security excellence, FishNet Security is the #1 provider of information security solutions that combine technology, services, support, and training. FishNet Security solutions have enabled 3,000 clients to better manage risk, meet compliance requirements, and reduce cost while maximizing security effectiveness and operational efficiency. For more information on FishNet Security, Inc., visit www.fishnetsecurity.com.